Compact Sensor System for Target Localization

Stian Tafjord Hovde

Thesis submitted for the degree of
Master in Cybernetics
30 credits

Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2017

Compact Sensor System for Target Localization

Stian Tafjord Hovde

Compact Sensor System for Target Localization

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

The main topic of this thesis is to design a compact sensor system that provides target localization functionality. The system should be low cost, small and lightweight, so it can be mounted on a standard assault rifle. To localize a given target, the system has to both determine its own position, and the distance and direction to the target. The most challenging part of this is to determine the heading.

To find this heading, our system uses a dual antenna Global Navigation Satellite System (GNSS) receiver. Because of the risk of GNSS signals being jammed, our proposed system also includes a stereo camera and an inertial measurement unit (IMU). Together these are able to maintain heading reference for a period of time after the loss of GNSS signals. The data from our sensors are fused in an error state Kalman filter implemented in Navigation Laboratory (NavLab).

Test results show that with the short antenna baseline available in this system, the GNSS receiver still gives a good heading estimate, if we compensate for receiver bias. Results also show that addition of the stereo camera reduces the heading drift by a factor of at least 2. In the worst case scenario, an acceptable heading is maintained for up to 5 minutes without GNSS, with some assumptions.

It is concluded that the designed sensor system is a viable device to extract target coordinates in the field. The results achieved in this thesis provide a good foundation for further work on this subject.

Preface

This thesis is the result of my master assignment, carried out at the end of my two year masters degree in cybernetics at the University of Oslo. The work was carried out between 24.01.2017 and 30.05.2017 at the Department of Technology Systems (ITS) at Kjeller.

I would like to thank my supervisors at the Norwegian Defence Research Establishment (FFI): Anders Rødningsby and Ørnulf Kandola, for giving me the subject of this thesis, and their help with completing it. I would also like to thank my supervisor at ITS, Oddvar Hallingstad.

I would also especially like to thank Thomas Olsvik Opsahl and Atle Skaugen at FFI for their help with the camera setup and field testing, and to Kjetil Bergh Ånonsen at FFI for his help with NavLab.

Kjeller, May 30, 2017

Stian Tafjord Hovde

Contents

List of Figures

List of Tables

Listings

Nomenclature

CEP Circular Error Probable

ESKF Error State Kalman Filter.

FFI Norwegian Defence Research Establishment (in Norwegian: Forsvarets forskningsinstitutt).

FOV Field Of View

GLONASS Global Navigation Satellite System (in Russian: Globalnaja Navigatsionnaja Sputnikovaja Sistema). Russian GNSS.

GNSS Global Navigation Satellite System.

GPS Global Positioning System. American GNSS.

IMU Inertial Measurement Unit.

INS Inertial Navigation System

ITS Department of Technology Systems (in Norwegian: Institutt for Teknologisystemer).

MEMS Micro Electro Mechanical System

NavLab Navigation Laboratory

ROS Robotic Operating System.

SBAS Satellite-Based Augmentation System

SLAM Simultaneous Localization And Mapping

VO Visual Odometry

Chapter 1

Introduction

1.1 Motivation

In the armed forces there is a need for soldiers in the field to be able to localize targets and determine target coordinates when requesting indirect fire support. Most of these soldiers would have target localization as a secondary role, as opposed to soldiers within artillery or mortar divisions.

Today, these second line target localizers are not equipped with advanced fire control systems to determine target coordinates, but instead rely on maps to estimate the latitude, longitude and height of the target. It is therefore a need to provide these soldiers with equipment that can be used to determine target coordinates in a fast and precise way. On the other hand, giving the soldier more equipment to carry and control may be a disadvantage unless the equipment is well integrated.

It is therefore of interest to investigate if the equipment the soldier is already carrying can be modified to give this functionality, with minor increase in weight. The cost of the system should also reflect that the target localization in this case is a secondary task.

The suggestion proposed by the Norwegian Defence Research Establishment (FFI) was to equip a standard assault rifle with a dual antenna Global Navigation Satellite System (GNSS) receiver, placing one antenna in each end of the rifle, to provide an accurate position and heading estimate for the soldier. This system will be reliant on GNSS signals, and since jamming of these signals has been quite common in modern battlefields, a robust support system is required.

The resulting idea for this thesis is then to create a proof-of-concept version of such a system. In addition to the dual antenna GNSS, an inertial measurement unit (IMU) and a camera are used to provide navigation when GNSS is unavailable. A requirement for the system is that the total target localization error should not exceed 30 m circular error probable (CEP), so it needs to be determined at what ranges and under which conditions we are able to achieve this. Target localization in the field is mostly done up to a range of about 2000 meters. Another requirement is for the total system to not add more weight than a standard under-barrel grenade launcher, about 1.5 kg.

1.2 Purpose and Goals

The overall goal of this thesis is to create a proof-of-concept version of a compact sensor system that can localize a given target. It should be small and lightweight so as not to interfere with normal use when mounted on a standard assault rifle.

When determining the location of a target, there are four main parameters one needs to determine: ones own position, and the heading, elevation and distance to the target. The position, elevation and distance can easily be determined using GNSS, accelerometers and laser rangefinders respectively. As will be discussed later, the heading is not that easy to determine. The heading error is also one of the largest contributors to the total target localization error. The primary focus of the thesis will therefore be to estimate the heading as accurately as possible.

The individual goals can be roughly summed up in the following points:

1. Find a method of logging and synchronising the data from our different sensors.
2. Build a testing platform where all our sensors can be mounted, to do realistic tests in the field.
3. Fuse the data from our different sensors, to get an optimal heading estimate from our system.
4. Test the dual antenna GNSS receiver and see how good a heading estimate we can get with different baseline lengths, and how it reacts to jamming.
5. Test the complete system and analyse the data.

1.3 Outline

The thesis has the following structure:

Chapter 2: Background: Presents the theoretical background used in this thesis.

Chapter 3: Platform: Presents the testing platform constructed for this project, including hardware and software.

Chapter 4: Sensor Models: Description of the mathematical sensor models implemented in NavLab.

Chapter 5: Results: Description of the tests performed with our platform, and presentation of the results from these tests.

Chapter 6: Discussion: Discussion of the testing results, and recommendations for further work.

Chapter 7: Conclusion: Summary of goals achieved, and the final conclusion.

Appendix A: Navigation Equations: Appendix with the navigation equations and their error propagation used in NavLab.

Appendix B: Code Listings: Appendix with the Matlab and C++ code used in the thesis.

Chapter 2

Background

This chapter presents some of the theoretical background used in this thesis. The material presented here serves as an introduction on the topic, a deeper explanation can be found in the provided references.

2.1 Notation

The notation used in this thesis is based on [9]. The general notation is presented in Table 2.1.

Symbol	Description
a	Scalar: lower case letter
A	Coordinate frame: upper case letter
$\boldsymbol{R}$	Matrix: bold upper case letter
$\boldsymbol{x}$	Coordinate free/geometrical vector: bold lower case letter
$\boldsymbol{x}^A$	Vector decomposed/represented in a specific coordinate frame: bold lower case letter with the coordinate frame as a right superscript

Table 2.1: General notation used in the thesis.

In the physical world, we use vectors to describe quantities like position, angular velocity etc. These quantities relate one coordinate frame to another, and to make the quantity unique, the two frames in question are given as a right subscript, as seen in Table 2.2.

Name	Symbol	Description
Position vector	$\boldsymbol{p}_{AB}$	A vector whose length and direction is such that it extends from the origin of frame A to the origin of frame B.
Velocity vector	$\boldsymbol{v}_{\underline{A}B}$	The velocity of the origin of coordinate frame B relative to coordinate frame A. Underline shows that both the orientation and position of A matters.
Acceleration vector	$\boldsymbol{a}_{\underline{A}B}$	The acceleration of the origin of coordinate frame B relative to coordinate frame A. Underline shows that both the orientation and position of A matters.
Rotation matrix	$\boldsymbol{R}_{AB}$	A 3x3 direction cosine matrix describing the orientation of frame B relative to frame A.
Angular velocity	$\boldsymbol{\omega}_{AB}$	The angular velocity of frame B relative to frame A.
Error vector	$\boldsymbol{e}_{AB}$	A vector describing the difference between $\hat{\boldsymbol{R}}_{AB}$ and $\boldsymbol{R}_{AB}$, interpeted as the error in $\hat{\boldsymbol{R}}_{AB}$.

Table 2.2: Symbols used to describe basic relations between two coordinate frames [9].

Note that this notation follows the "rule of closest frames". This means that when transforming quantities between different coordinate frames, the "closest" frames in the equation must be the

same one (e.g. to get the rotation matrix $\boldsymbol{R}_{AC}$, you multiply $\boldsymbol{R}_{AB}$ with $\boldsymbol{R}_{BC}$. Here we see that the B frames are "closest").

The vectors in Table 2.2 are written in a coordinate free form. Before implementation on a computer, they must be decomposed in a selected coordinate frame (e.g. the position vector $\boldsymbol{p}_{AB}$ decomposed in frame C is $\boldsymbol{p}^{C}_{AB}$).

Until now, three arbitrary frames A, B and C has been used as an example. In the main part of this thesis the used coordinate frames will have a meaning given by Table 2.3

Coordinate frame	Description
I	Inertial space
E	Earth-fixed coordinate frame, moves and rotates with the Earth. The x-axis is along the Earth's rotation axis, pointing north (the yz-plane coincides with the equatorial plane), the y-axis points towards longitude +90°(east).
B	Body-fixed coordinate frame, attached to the navigating vehicle. The x-axis points forward, the y-axis to the right (starboard) and the z-axis in the vehicle's down direction.
L	Local coordinate frame, with the origin directly above or below B, at Earth's surface. The z-axis is pointing down. Initially, the x-axis points towards north, and the y-axis points towards east, but as the vehicle moves they are not rotating about the z-axis (their angular velocity relative to the Earth has zero component along the z-axis).

Table 2.3: Coordinate frames in use [9].

2.2 Northfinding

As discussed in the introduction, finding an accurate heading estimate is one of the most important goals for our system. The reason for this can be explained by looking at Figure 2.1.

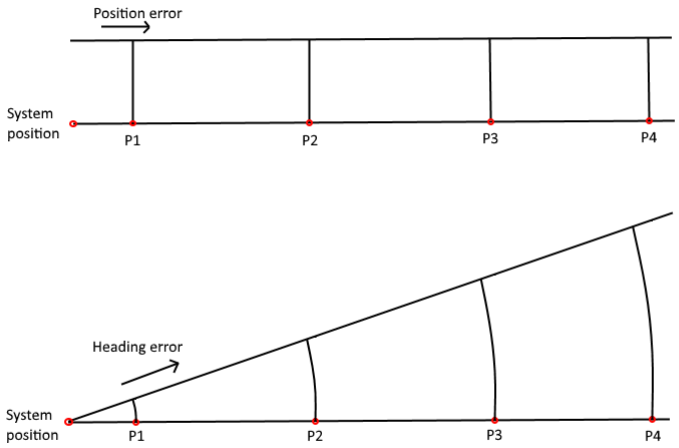

Figure 2.1: Investigating the position error caused by heading error.

Assume our system is standing at the "system position" mark. We can see that any position error we have for our starting position will lead to a constant error in our estimation of the targets position, no matter the distance to our target. On the other hand, we can see that a heading error

will cause target position error to increase based on the distance. For example, a 1 degree heading error will at 1 km cause a target position error of 17.5 m, while at 2 km this increases to 35 m.

It is therefore obvious that an accurate estimate of our systems heading is vital for us to get an accurate estimate of the position of the target we are trying to localize.

There are seven main different ways to determine the heading of an object [10]. For convenience these are listed in Table 2.4.

Method	Vector in use	Notes
1. Magnetic compass	$\boldsymbol{m}_B$	Increasing latitude decreases accuracy
2. Gyrocompass	$\boldsymbol{\omega}_{IE}$	Increasing latitude decreases accuracy
3. Observing multiple external objects	$\boldsymbol{p}_{O_1O_2}$	
4. Measure bearing to object with known position	$\boldsymbol{p}_{BO}$	GNSS required
5. Multi-antenna GNSS	$\boldsymbol{p}_{B_1B_2}$	GNSS required
6. Vehicle velocity	$\boldsymbol{v}_{\underline{E}B}$	GNSS required Vehicle motion required
7. Vehicle acceleration	$\boldsymbol{a}_{\underline{E}B}$	GNSS required Vehicle motion required

Table 2.4: Summary of the methods to find heading [10].

We can now walk through the list of methods and see which ones are suitable to find the heading for our system:

- **Method 1**: No; a magnetic compass would be sensitive to magnetic materials in the ground, on our rifle, and attached to the soldier, and would therefore be too inaccurate for our purpose.

- **Method 2**: No; a gyrocompass that gives us an accurate enough estimate of the heading would be too heavy and too expensive.

- **Method 3**: Sometimes; can be used when multiple recognisable objects can be seen (e.g. celestial objects during clear weather).

- **Method 4**: Sometimes; can be used when a recognisable object can be seen.

- **Method 5**: Maybe; it is uncertain whether the baseline length between antennas achievable on a soldiers rifle is long enough to give a good heading estimate.

- **Method 6**: Maybe; it is uncertain if the velocity exerted by a soldier is high enough to give a good heading estimate.

- **Method 7**: Maybe; it is uncertain if the acceleration exerted by a soldier is high enough to give a good heading estimate.

As we can see, there is not a simple way to give an accurate heading estimate with our system in all situations. As method 3 and 4 are only available when recognisable objects are spotted, we will not be using these in our thesis. Also, since we are focused on getting an accurate heading measurement, our system will not be in translational motion during the testing. We will therefore not be able to measure any velocity or acceleration, and will thus not investigate method 6 or 7. This leaves us with method 5, a multi-antenna GNSS system, which we will be looking at more thoroughly in this thesis.

The drawback with the multi-antenna GNSS method, is that heading accuracy depends on the length of the baseline between the antennas. The maximum baseline we can have on a standard assault rifle is around 0.5 meters. It needs to be investigated if this baseline is sufficient, or if the baseline available to us on the rifle has to be extended by some means.

2.3 Rotation Matrix

The material in this section is based on [12] and [6]. A rotation matrix is a 3x3 direction cosine matrix that describes the orientation of one frame in relation to another. One way to formulate a general rotation matrix $\boldsymbol{R}_{AB}$ is the following: The coordinate frame A is first rotated around its own x-axis with an angle of θ_1, then an angle θ_2 around its new y-axis, and finally an angle θ_3 around its new z-axis. The coordinate frame we end up with after all these rotations is B. The rotation matrix can therefore be expressed as:

$$\begin{aligned}\boldsymbol{R}_{AB} &= \begin{bmatrix}1 & 0 & 0\\ 0 & c(\theta_1) & -s(\theta_1)\\ 0 & s(\theta_1) & c(\theta_1)\end{bmatrix}\begin{bmatrix}c(\theta_2) & 0 & s(\theta_2)\\ 0 & 1 & 0\\ -s(\theta_2) & 0 & c(\theta_2)\end{bmatrix}\begin{bmatrix}c(\theta_3) & -s(\theta_2) & 0\\ s(\theta_2) & c(\theta_3) & 0\\ 0 & 0 & 1\end{bmatrix}\\ &= \begin{bmatrix}c(\theta_2)c(\theta_3) & -c(\theta_2)s(\theta_3) & s(\theta_2)\\ s(\theta_1)s(\theta_2)c(\theta_3)+c(\theta_1)s(\theta_3) & -s(\theta_1)s(\theta_2)s(\theta_3)+c(\theta_1)c(\theta_3) & -s(\theta_1)c(\theta_2)\\ -c(\theta_1)s(\theta_2)c(\theta_3)+s(\theta_1)s(\theta_3) & c(\theta_1)s(\theta_2)s(\theta_3)+s(\theta_1)c(\theta_3) & c(\theta_1)c(\theta_2)\end{bmatrix}\end{aligned} \tag{2.1}$$

Here $s(\cdot)$ is the sine function and $c(\cdot)$ is the cosine function. The three parameters θ_1, θ_2 and θ_3 give a unique representation of $\boldsymbol{R}_{AB}$, and we can therefore organise these in a vector:

$$\boldsymbol{\theta}_{AB} = \begin{bmatrix}\theta_1\\ \theta_2\\ \theta_3\end{bmatrix} \tag{2.2}$$

Some key characteristics of the rotation matrix follows:
To transform a vector represented in frame B to frame A we have that:

$$\boldsymbol{p}^A_{AB} = \boldsymbol{R}_{AB}\boldsymbol{p}^B_{AB} \tag{2.3}$$

To create a new rotation matrix $\boldsymbol{R}_{AC}$:

$$\boldsymbol{R}_{AC} = \boldsymbol{R}_{AB}\boldsymbol{R}_{BC} \tag{2.4}$$

Since we assume all our coordinate frames have orthonormal basis vectors, the rotation matrix will be an orthogonal matrix:

$$\boldsymbol{R}_{AB}(\boldsymbol{R}_{AB})^T = \boldsymbol{I} \tag{2.5}$$

If we look at the change in the rotation matrix over time, we can find the time derivative of equation 2.5:

$$\dot{\boldsymbol{R}}_{AB}(\boldsymbol{R}_{AB})^T + \boldsymbol{R}_{AB}(\dot{\boldsymbol{R}}_{AB})^T = \frac{d}{dt}\boldsymbol{I} = 0 \tag{2.6}$$

Equation 2.6 can be written:

$$\dot{\boldsymbol{R}}_{AB}(\boldsymbol{R}_{AB})^T + (\dot{\boldsymbol{R}}_{AB}(\boldsymbol{R}_{AB})^T)^T = 0 \tag{2.7}$$

If we now define

$$\boldsymbol{S} = \dot{\boldsymbol{R}}_{AB}(\boldsymbol{R}_{AB})^T \tag{2.8}$$

we have that:

$$\boldsymbol{S} + \boldsymbol{S}^T = 0 \tag{2.9}$$

This means that $\boldsymbol{S}$ is a skew symmetric matrix, and there exists a vector $\boldsymbol{\omega}^B_{AB} = [\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_3]^T$ that fulfils:

$$\boldsymbol{S}(\boldsymbol{\omega}^B_{AB}) = \begin{bmatrix}0 & -\boldsymbol{\omega}_3 & \boldsymbol{\omega}_2\\ \boldsymbol{\omega}_3 & 0 & -\boldsymbol{\omega}_1\\ -\boldsymbol{\omega}_2 & \boldsymbol{\omega}_1 & 0\end{bmatrix} \tag{2.10}$$

We interpret this vector as the angular velocity of the A system as seen from the B system. It can also be shown that the cross product operator can be written on the following form [12]:

$$\boldsymbol{\omega}_{AB}^{B} \times \boldsymbol{e}_{AB}^{B} = \boldsymbol{S}(\boldsymbol{\omega}_{AB}^{B})\boldsymbol{e}_{AB}^{B} \tag{2.11}$$

By multiplying $\boldsymbol{R}_{AB}$ from the right with equation 2.6 we end up with:

$$\dot{\boldsymbol{R}}_{AB} + \boldsymbol{R}_{AB}(\dot{\boldsymbol{R}}_{AB})^{T}\boldsymbol{R}_{AB} = 0 \tag{2.12}$$

using 2.8 we get:

$$\dot{\boldsymbol{R}}_{AB} = \boldsymbol{R}_{AB}(\dot{\boldsymbol{R}}_{AB})^{T}\boldsymbol{R}_{AB} = -\boldsymbol{S}^{T}\boldsymbol{R}_{AB} = 0 \tag{2.13}$$

Thus, we end up with:

$$\dot{\boldsymbol{R}}_{AB} = \boldsymbol{S}(\boldsymbol{\omega}_{AB}^{A})\boldsymbol{R}_{AB} = \boldsymbol{R}_{AB}\boldsymbol{S}(\boldsymbol{\omega}_{AB}^{B}) \tag{2.14}$$

It is also useful to know how a rotation matrix that is approximately equal to $\boldsymbol{R}_{AB}$, denoted as $\hat{\boldsymbol{R}}_{AB}$, can be expressed. From [6], we have that:

$$\hat{\boldsymbol{R}}_{AB} \approx (\boldsymbol{I} + \boldsymbol{S}(\boldsymbol{e}_{AB}^{A}))\boldsymbol{R}_{AB} = \boldsymbol{R}_{AB}(\boldsymbol{I} + \boldsymbol{S}(\boldsymbol{e}_{AB}^{B})) \tag{2.15}$$

Where $\boldsymbol{e}_{AB}^{A}$ is the vector that describes the rotation needed to correct the approximate rotation matrix $\hat{\boldsymbol{R}}_{AB}$ to the true rotation matrix $\boldsymbol{R}_{AB}$.

If we now assume that any rotation matrix we calculate can be written as the true rotation matrix plus some error: $\hat{\boldsymbol{R}}_{AB} = \boldsymbol{R}_{AB} + \delta\boldsymbol{R}_{AB}$, it is clear that:

$$\delta\boldsymbol{R}_{AB} = \boldsymbol{S}(\boldsymbol{e}_{AB}^{A})\boldsymbol{R}_{AB} = \boldsymbol{R}_{AB}\boldsymbol{S}(\boldsymbol{e}_{AB}^{B}) \tag{2.16}$$

2.4 Stochastic Variables

The material in this section is based on [4]. A stochastic, or random, variable X is in it simplest terms a variable which is assigned values at random. It can be described as a function of the outcomes of some random experiment. The probability for the stochastic variable to take a given value is defined by the probability distribution function $F(x)$:

$$F(x) = p(X \leq x) \tag{2.17}$$

or the probability density function $f(x)$:

$$f(x) = \frac{dF(x)}{dx} \tag{2.18}$$

The relation between these are:

$$F(x) = \int_{-\infty}^{x} f(u)du \tag{2.19}$$

A stochastic variable has an expectation $\bar{x}$ and a variance σ^2 defined as:

$$\bar{x} = E\{X\} = \int_{-\infty}^{\infty} xf(x)dx \tag{2.20}$$

$$\sigma^2 = Var\{X\} = \int_{-\infty}^{\infty} (x - E\{X\})^2 f(x)dx = E\{(X - \bar{x})^2\} \tag{2.21}$$

The expectation, or mean value, is the average value a number of observations of X tend towards, as the number of observations increases. The variance of a random variable is the mean squared deviation of the random variable from its mean. The square root of the variance is known as the standard deviation.

If a set of stochastic variables are independent, it holds that the variance of the sum of those stochastic variables is equal to the sum of the variances:

$$\sigma_{x}^{2} = \sum_{i=1}^{n} \sigma_{x_i}^{2} \tag{2.22}$$

Given two stochastic variables X and Y, the variance between these can be found. This is known as the covariance:

$$Cov\{X,Y\} = E\{(X-\bar{x})(Y-\bar{y})\} \tag{2.23}$$

Throughout this thesis we will be using normal-distributed, or Gaussian, stochastic variables. If a variable X is Gaussian it is denoted as $X \sim \mathcal{N}(\bar{x}, \sigma^2)$. The probability density function of a Gaussian variable is given as:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\frac{(x-\bar{x})^2}{\sigma^2}} \tag{2.24}$$

2.5 1. Order Markov Process

The material in this section is based on [4]. In Chapter 4 it is shown that our measurements is affected by coloured noise, also referred to as a bias, in addition to white noise. Modelling this bias as a 1. order Markov process has from experience at FFI shown to give a good model of the real noise.

A continuous process $x(t)$ is a 1. order Markov process if for every k, and for $t_1 < t_2 < ... < t_k$, it is true that:

$$F[x(t_k)|x(t_{k-1}), ..., x(t_1)] = F[x(t_k)|x(t_{k-1})] \tag{2.25}$$

That is, the probability distribution function for $x(t_k)$ is only dependent on the value one time step in the past. If the probability distribution function is dependent on values n time steps in the past, it is called a n-th order Markov process.

If the continious process $x(t)$ is a 1. order Markov process, it can be formulated as the differential equation:

$$\frac{dx}{dt} + \frac{1}{T}x = \gamma \tag{2.26}$$

Rearranging Equation 2.26 yields:

$$\dot{x} = -\frac{1}{T}x + \gamma \tag{2.27}$$

Where T is the time constant, and γ is the white noise which drives the bias. If the probability density function of γ is Gaussian, the process $x(t)$ is known as a 1. order Gauss-Markov process.

2.6 Error State Kalman Filter

One of our main goals is to find the best possible estimate of our platforms heading by fusing our sensor measurements in an optimal way. This is done using the error state Kalman filter (ESKF), also known as the indirect Kalman filter, in NavLab [8] (see Section 3.2.4).

We first look into how the standard Kalman filter works. The Kalman filter is an optimal estimator for linear stochastic systems with additive zero mean Gaussian noise, derived in for example [2]. The information in these systems is first specified as a continuous-discrete state-space model:

$$\dot{x}(t) = F(t)x(t) + L(t)u(t) + G(t)v(t) \tag{2.28}$$

$$y_k = H_k x_k + \xi_k \tag{2.29}$$

Equation 2.28 is called the process model, which models the transformation of the process state $x(t)$ at time t. Equation 2.29 is called the measurement model, which describes the relationship between the process state x_k and the measurements y_k at time step k. F is the system matrix, L is the control matrix, G is the process noise matrix and H is the measurement matrix.

$v(t)$ and ξ_k is the process and measurement noise. These are white and uncorrelated, with spectral density matrix $\tilde{Q}$ and covariance matrix W_k respectively. $v(t)$, ξ_k and $x(t_0)$ is uncorrelated.

The Kalman filter is an recursive algorithm that combines sensor mesurements and the system information in an optimal way. Optimal here means that the algorithm finds a minimum variance estimate. Recursive means that the filter only need the estimated state and its covariance from the previous time step, and the current measurement, to get the estimate for the current state. The continuous-discrete Kalman filter algorithm is given in [2]:

$$\begin{aligned} \bar{x}_0 &= E(x_0) \\ \bar{P}_0 &= E[(x_0 - \bar{x}_0)(x_0 - \bar{x}_0)^T] \end{aligned} \tag{2.30}$$

$$\begin{aligned} \dot{\bar{x}}(t) &= F(t)\bar{x}(t) + L(t)u(t) \\ \dot{\bar{P}}(t) &= F(t)\bar{P}(t) + \bar{P}(t)F^T(t) + G(t)\tilde{Q}G^T(t) \end{aligned} \tag{2.31}$$

$$\begin{aligned} K_k &= \bar{P}_k H_k^T (H_k \bar{P}_k H_k^T + W_k)^{-1} \\ \hat{x}_k &= \bar{x}_k + K_k(y_k - H_k \bar{x}_k) \\ \hat{P}_k &= (I - K_k H_k)\bar{P}_k \end{aligned} \tag{2.32}$$

Before implementation into a computer, the process model needs to be discretized. As NavLab handles this internally, this procedure will not be listed here, but is well explained in [6]. We then end up with the discrete process model:

$$x_{k+1} = \Phi_k x_k + \Lambda_k u_k + \Omega_k v_k \tag{2.33}$$

where Φ, Λ and Ω are the discretized versions of matrices F, L and G respectively. We now end up with the discrete Kalman filter [2]:

$$\begin{aligned} \bar{x}_0 &= E(x_0) \\ \bar{P}_0 &= E[(x_0 - \bar{x}_0)(x_0 - \bar{x}_0)^T] \end{aligned} \tag{2.34}$$

$$\begin{aligned} \bar{x}_{k+1} &= \Phi_k \hat{x}_k + \Delta_k u_k \\ \bar{P}_{k+1} &= \Phi_k \hat{P}_k \Phi_k^T + \Omega_k Q_k \Omega_k^T \end{aligned} \tag{2.35}$$

$$\begin{aligned} K_k &= \bar{P}_k H_k^T (H_k \bar{P}_k H_k^T + W_k)^{-1} \\ \hat{x}_k &= \bar{x}_k + K_k(y_k - H_k \bar{x}_k) \\ \hat{P}_k &= (I - K_k H_k)\bar{P}_k \end{aligned} \tag{2.36}$$

Here Equations 2.34 is the initialization step, where the a priori information of the state $\bar{x}_0$ and its covariance $\bar{P}_0$ are input. Equations 2.35 is the prediction step, which uses the estimated state from the previous time step to calculate an estimated state at the current time step. This is also known as the a priori state estimate. Finally, Equations 2.36 is the update step, where the predicted state estimate is refined based on measurements. This is known as the a posteriori state estimate.

Typically, the prediction and update steps alternate, with the prediction step bringing the state estimate to the next time step, and the update step refining it with information from the measurements. However, if the measurements are missing in some time steps, the update step is skipped, and several prediction steps can be performed instead.

Note that during calculations in a computer, numerical errors might arise that makes the covariance matrix of the state vector not symmetrical and positive semidefinite. In these cases, the Kalman filter will not be optimal. Therefore, the covariance update equation in NavLab is written on what is called the Joseph form [13]:

$$\hat{P}_k = (I - K_k H_k)\bar{P}_k(I - K_k H_k)^T + K_k R_k K_k^T \tag{2.37}$$

Since we have measurements of the states we are interested in, it is sufficient to estimate the error in these. The states in the Kalman filter then become error states, and we end up with the error state Kalman filter (ESKF). These error states are calculated using the difference between the inertial navigation system (IMU measurements and navigation equations) and external sources of data (e.g. GNSS, camera).

Some advantages using the ESKF over a standard Kalman filter in conjunction with inertial navigation systems (INS) are [22]:

- The standard Kalman filter using the total state representation has to incorporate a dynamic model of the vehicle, as well as attempt to suppress noisy and erroneous data at a relatively high frequency. Using the ESKF the INS is able to follow the high frequency motions very accurately, so there is no need to model these dynamics explicitly in the filter.
- The dynamics involved in the total state description include a high frequency component and is only well described by a non-linear model. The dynamics upon which the ESKF is based are a set of inertial system error propagation equations which are low frequency and very adequately represented as linear.
- If the standard Kalman filter fails, the entire navigation algorithm fails. The ESKF can in case of a failure continue to provide estimates by acting as an integrator on the INS data.
- Because the ESKF is outside the INS loop and based on low frequency dynamics, its sampling rate can be much lower than a standard Kalman filter.

For these reasons, the ESKF is almost always used in terrestrial aided inertial navigation systems.

2.7 Camera Calibration

Camera calibration is needed to find the intrinsic parameters of the specific camera that is being used. The calibration process also outputs the lens distortion parameters that we have to correct for, before running the visual navigation algorithms.

The common way to accomplish this is to take multiple pictures with our cameras of a chess board with known size. This is known as Zhang's method [26]. The pictures should be taken from several different angles, and with varying size and rotation. Feeding these images into the calibration algorithm in for example OpenCV (see Section 3.2.2) or the `camera_calibration` package in ROS (see Section 3.2.1), will ouput the intrinsic matrix $\boldsymbol{K}$ and the distortion coefficients:

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \text{ distortionCoefficients} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{2.38}$$

In the intrinsic matrix, f_x and f_y are the camera focal lengths in pixels, c_x and c_y is the camera's principal point (or optical centre) in pixels, and s is the skew coefficient, which is normally set to 0 since the image axes are usually perpendicular.

The distortion coefficients k_1, k_2 and k_3 describe the radial distortion in the camera that manifests in form of the "barrel" or "fish-eye" effect. p_1 and p_2 describe the tangential distortion, which occurs when the image taking lenses are not perfectly parallel to the imaging plane.

In the case of stereo cameras, the calibration procedure will also output the rotation and translation between the two cameras.

Now that we have the intrinsic matrix and the distortion coefficients, we are able to rectify our images in for example OpenCV (see section 3.2.2), before sending them to the visual navigation algorithms.

Chapter 3

Platform

This chapter details the specific hardware and software used on our testing platform. It also shows the complete system, errors in mounting, and how the coordinate frames of the different components relate to each other.

3.1 Hardware

3.1.1 GNSS

As explained in the introduction, we will be using a GNSS receiver with dual antenna input to measure the heading of our system accurately. The FlexPak6D from NovAtel has been used in this project. This is a light and compact receiver unit with several connection ports for easy access. The FlexPak6D tracks both GPS and GLONASS signals, as well as satellite-based augmentation system (SBAS) signals that may improve GNSS accuracy.

Figure 3.1: FlexPak6D Dual Antenna GNSS Receiver [17].

Some key specifications of the FlexPak6D is listed in Table 3.1. These specifications are acquired from the FlexPak6D user manual [17]. The manual does not specify what kind of accuracy measure that are used for the different accuracy parameters. We will therefore assume they are given as standard deviations.

As we can see from Table 3.1, the accuracy of the heading measurement is only listed with baselines of 4 and 2 meters. Since our baseline will be much shorter than 2 meters, we have to test how much this will affect heading accuracy.

The FlexPak6d can output different data sets depending on user need. The full list of different log types can be found in the firmware reference manual [18]. We will use the *bestpos* log which

Name	Value	Unit
Primary and secondary RF	GPS: L1/L2 GLONASS: L1/L2	[]
Other signals	SBAS	[]
Horizontal position accuracy L1/L2	1.2	[m]
Heading accuracy 2m baseline	0.08	[deg]
Heading accuracy 4m baseline	0.05	[deg]
Size	147 x 113 x 45	[mm]
Weight	315	[g]

Table 3.1: NovAtel FlexPak6D specifications [17].

outputs position (longditude, latitude and height), and the *heading* log which outputs platform heading relative to true north.

Heading is determined by measuring signal phase difference between the two antennas from each satellite in view. The measurements are then used to calculate the vector between the reference antenna and the rover antenna. In this project the rear antenna is used as reference, so that the vector points forwards. To determine heading, the receiver uses the NovAtel ALIGN Heading and Relative Positioning GNSS Firmware [15].

To acquire the logs from the receiver, NovAtel has its own windows-based GUI called NovAtel Connect, that gives easy access the receiver via USB, to log data or change the configuration. We will also use a ROS software package to connect to the receiver and log data (see Section 3.2.1)

Another point to note is that the rest of our system is using Unix time (or Unix epoch time), while our GNSS receiver outputs GPS time in GPS week and seconds into GPS week. We therefore have to do a quick conversion to Unix time.

Unix time started at 1.1.1970 UTC, while GPS time started at 6.1.1980 UTC. This difference turns out to be 315964800 seconds. In addition, as of May 2017, 18 leap seconds has been added since the introduction of GPS time. There are also 604800 seconds in a week. Our formula to convert from GPS time to Unix time then becomes:

$$unixTime = gpsWeek \cdot 604800 + gpsSecond + 315964800 - 18 \tag{3.1}$$

In addition we also acquired two antennas of the type 42GOXX16A4-XT-1-1-Cert from NovAtel. These are compact antennas with dual frequency support, and combined GPS + GLONASS signal reception. Some key specifications of our antennas is listed in Table 3.2. These specifications are acquired from NovAtel's antennas brochure [16].

Figure 3.2: 42GOXX16A4-XT-1-1-Cert Antenna [16].

Name	Value	Unit
Signals received	GPS: L1/L2 GLONASS: L1/L2 BeiDou: B1/B2 SBAS	[]
Size	119.38 x 76.20 x 22.80	[mm]
Weight	227	[g]

Table 3.2: NovAtel 42GOXX16A4-XT-1-1-Cert specifications [16].

3.1.2 IMU

An IMU is a device that uses gyroscopes and accelerometers to measure body angular rate and specific force. These measurements are integrated by the navigation equations listed in Appendix A, which outputs the position, orientation and velocity.

The IMU we will be using is the MTi-100 from Xsens Technologies B.V. The MTi-100 is a micro electro mechanical system (MEMS) based IMU that contain a three-axis gyroscope and a three-axis accelerometer. It also contains a magnetometer and a barometer, although we at present do not plan to use measurements from these sensors. This IMU was chosen for its small size and low cost, and also based on recommendations from other projects at FFI.

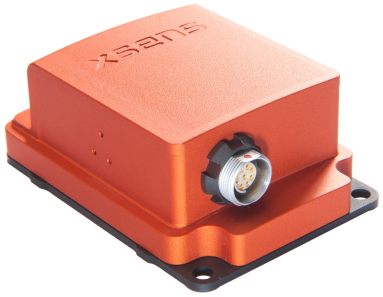

Figure 3.3: Xsens MTi-100 IMU [24].

Some key specifications of the MTi-100 is listed in Table 3.3. These specifications are acquired from the MTi-100 series user manual [24], and the calibration certificate received with the MTi-100. The manual and calibration certificate does not specify what kind of accuracy measure that are used for the different accuracy parameters. We will therefore assume they are given as standard deviations.

Name	Value	Unit
Gyro full range	450	[deg/s]
Gyro bias	0.5	[deg/s]
Gyro measurement noise density	0.015	[deg/s/$\sqrt{\text{Hz}}$]
Accelerometer full range	50	$[m/s^2]$
Accelerometer bias	0.03	$[m/s^2]$
Accelerometer measurement noise density	80	[μg/$\sqrt{\text{Hz}}$]
Size	57 x 42 x 24	[mm]
Weight	55	[g]

Table 3.3: Xsens MTi-100 specifications [24].

Xsens also has its own logging program for its MTi platforms, called MT Manager. This runs on the Windows OS and allows both logging of data and configuration of the sensors. Unfortunately, this program only logs the start time to the nearest minute, which is too inaccurate for our purpose. We have therefore used ROS to log data from the IMU (see Section 3.2.1).

3.1.3 Camera

Two different cameras have been used during this project. An inexpensive monocular webcam was used in the beginning to test out different functions and algorithms, while for the testing phase, a stereo camera was used.

The switch from a mono to stereo camera was done because the different visual navigation algorithms cannot estimate motion or position on a metric scale with a monocular camera. The estimate becomes relative to some unknown scaling factor. This scale could be found using an IMU to estimate the motion carried out between each frame, but since we will be stationary during our tests, this will not work. With a stereo camera, we find the scale directly because we know the distance between the two camera centres, and can therefore triangulate points in the scene.

The stereo camera rig consists of two cameras of the type Blackfly S Color 5.0 MP USB3 Vision (Sony IMX250), and two Xsens MTi-100 IMU's, which are described in Section 3.1.2. A picture of the rig can be seen in Figure 3.4. This rig was previously built by FFI, and was made available in order to take synchronized image pairs during the testing of our system. The cameras are capable of capturing images at a rate up to 70 Hz.

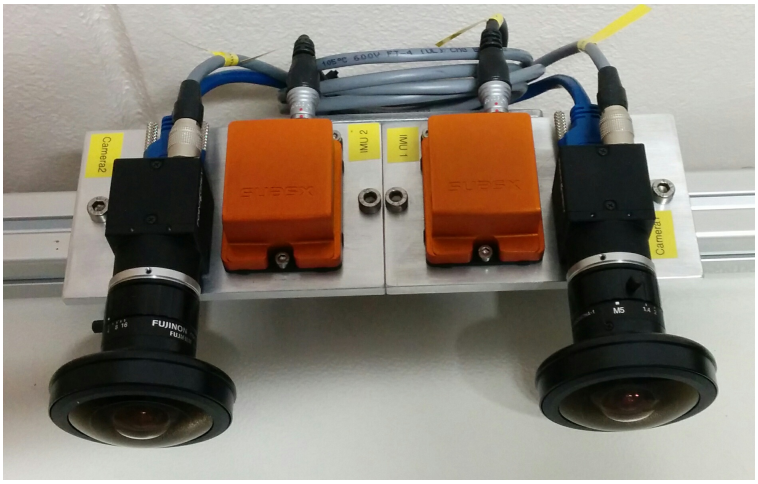

Figure 3.4: Stereo camera and IMU rig.

This camera type uses a global shutter, as opposed to a rolling shutter, to capture images. This means that all photosites receive light at the same time, as opposed to row by row with a rolling shutter. The effects of a rolling shutter can best be seen e.g. when photographing the propeller of a plane.

The cameras are fitted with fisheye lenses, providing a near 180° field of view (FOV). This makes them suited for visual navigation operations, since the image pairs from one time step to the next need to overlap in order for the algorithm to recognise feature points between them. At the same time, an increase in FOV lowers the pixel per degree resolution in the image, making the visual navigation algorithms more susceptible to noise.

In order to capture the images, the rig is connected to its own computer running the Ubuntu Linux OS. The images are stored in .bin files, and the software to extract these was provided by FFI as Matlab code. After the extraction the images have been cropped to 1200 by 1200 pixels. We can now calculate the new FOV using the following equations [1]:

$$fov_y = 2\tan^{-1}\left(\frac{h}{2f_y}\right) = 2\tan^{-1}\left(\frac{1200}{2\cdot 519.1}\right) = 98.27^\circ \tag{3.2}$$

$$fov_x = 2\tan^{-1}\left(\frac{w}{2f_x}\right) = 2\tan^{-1}\left(\frac{1200}{2\cdot 519.4}\right) = 98.24^\circ \tag{3.3}$$

Here h and w is the height and width of the image in pixels, f_y and f_x is the focal lengths found during the calibration procedure (see Section 2.7).

To time stamp the image pairs, the cameras send a sync pulse to FFI's DSU logger, which pairs each pulse with time information from a GPS receiver. The time log is then stored, and can be put together with the image data at a later time. Because of the risk of GPS signals being jammed,

this method will not be acceptable for the final product. For further work, another way must be implemented to synchronise images to other data, for example by using ROS (see Section 3.2.1). The Matlab code to extract these time logs was also made available by FFI.

3.2 Software

3.2.1 ROS

Robotic Operating System (ROS) is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [21]. A ROS system consists of a number of independent packages, that communicate with each other using a publisher/subscriber system. For example, one of our sensors drivers can publish the sensor data on a topic, which can then be read by any number of subscribers. ROS is used in this project in order to easily connect to our different sensors, read the data they output and synchronise them. ROS is developed for the Ubuntu Linux OS.

To connect to our IMU we use the `xsens_driver` package available in the ROS library. This package reads the data transmitted by the IMU through USB, and publishes it on a ROS message topic. The raw IMU data can now be sent to a filter, like the `imu_filter_madgwick` package, to calculate the orientation. As we are sending the raw IMU data directly into NavLab, this part is not implemented in the final result.

For our GNSS receiver we had to write a new package, since this did not exist. NovAtel does provide a Linux USB driver though, so we are able to easily communicate with the receiver. The package written is shown in Listing B.3. This package opens up communications with the receiver, then transmits the commands for the receiver to start logging the data we need. It then receives this data and publishes it on a ROS message topic.

In addition we have also run some tests of different ROS packages using our monocular webcam. We start by reading images from the camera using the `cv_camera` package. We then run a calibration sequence as detailed in Section 2.7 with the `camera_calibration` package. To use the data from the calibration and rectify our images, we use the `image_proc` package. We tried to send the images to the `viso2_ros` package, which is the ROS wrapper for LIBVISO2 as explained in Section 3.2.3, to output odometry data. Due to the limitations of the monocular version of LIBVISO2, the resulting data was not used further. Since we used the stereo camera rig for our testing, this part is not implemented in the final result.

An idea for future expansion into a real-time system is to send all the data we have now accumulated to the `robot_localization` package. This package accepts a multitude of sensor information, and combines them using an extended Kalman filter to give an estimate to the position of the object in question. As we are doing our estimation in NavLab, this has not been tested with our sensor data.

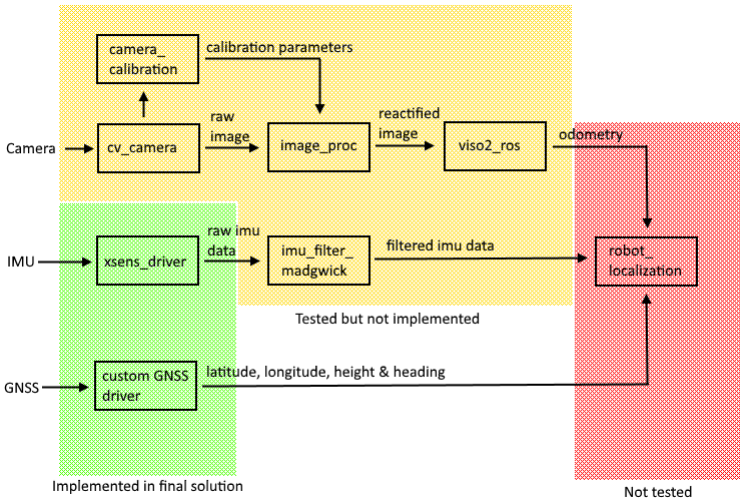

Figure 3.5: ROS implementation.

Figure 3.5 shows an overview of the packages we have used in ROS. It also shows the messages being published and subscribed by the different packages, and what parts that are implemented in the final result.

Once all the data is being published on the different ROS topics, we can start logging them into a ROS bag file. The bag file format stores the ROS messages we want, and allows us to replay them whenever we need. This way we can quickly and easily store data out in the field, and then process it afterwards.

For future work it might be useful to use the Robotic Toolbox available in Matlab. This toolbox has the functionality to do real time reading of ROS topics. It also has tools for working with ROS bag files.

3.2.2 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library [19]. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.

We will use a few of the OpenCV functions to work on our images before they are sent into the visual navigation algorithms. The OpenCV functions can be seen in action in Listing B.2, denoted with the cv namespace. As we are working on pictures from cameras with fisheye lenses, we must also use the fisheye namespace where applicable.

The OpenCV library is also available in Matlab, using the mex functions in mexopencv [25]. An example of this in use can be seen in Listing B.1. As these mex functions do not contain the fisheye namespace, the rectification maps needed are directly imported from the C++ script.

The following OpenCV functions have been used:

stereoRectify: Inputs the intrinsic matrix, distortion coefficients and the rotation and translation between the cameras found in the calibration procedure (see Section 2.7). Outputs the rectification transforms for each camera. These are two rotation matrices that that virtually make both camera image planes the same plane. It also outputs a projection matrix for each camera that transform points from the world frame into the new coordinates. Note that in OpenCV 3.2, fisheye::stereoRectify does not work properly. Therefore the results from this function is imported directly from OpenCV 3.1.

initUndistortRectifyMaps: Inputs the intrinsic matrix, distortion coefficients and the rectification transform and projection matrix found in stereoRectify. Outputs the undistortion and rectification transformation map for each camera. These maps are used by the remap function to correct for the distortion, and transforms the images such that the epipolar lines on both images become horizontal and have the same y-axis coordinate. Horizontal epipolar lines helps the visual navigration algorithms to find feature points between the image pair.

imread: Loads an image from the specified file and returns it.

remap: Transforms the source image using the specified map.

equalizeHist: Equalizes the histogram of a grayscale image. This normalizes the brightness and increases the contrast of the image, making it easier for the visual navigration algorithm to find feature points.

medianBlur: Blurs the image using a median filter. Done to reduce the noise in the image.

3.2.3 Visual Navigation Algorithms

To get useful measurements from the camera, we need to run the pictures through an algorithm that is able to estimate and output the motion the camera experiences. Two different algorithms has been tested, and are detailed here.

LIBVISO2

LIBVISO2 (Library for Visual Odometry 2) is a very fast cross-platfrom C++ library with MATLAB wrappers for computing the 6 DOF motion of a moving mono/stereo camera [11]. A ROS wrapper for LIBVISO2 also exist, that reads directly from a ROS image topic and outputs odometry data.

LIBVISO2 estimates the translation and rotation of either a monocular or stereo camera by extracting and matching corner-like features between two consecutive images. This means its

important for our images to overlap between time steps in order for the algorithm to recognise the same feature points.

Additionally, LIBVISO2 uses an iterated sigma point Kalman filter to cope with non-linearities in the measurement equation. It also uses a random sample consensus outlier rejection scheme to detect and remove false matches, and features located on independently moving objects.

To run LIBVISO2 with our monocular webcam, we use the ROS wrapper in the `viso2_ros` package. Due to the monocular version of LIBVISO2 assuming that the camera is moving at a known and fixed height over the ground, this run did not give any usable data.

The Matlab script used to run LIBVISO2 with our stereo images is shown in Listing B.1. This is a modification of the stereo example script that is included with the Matlab wrappers. Before sending our images through the algorithm, they are rectified by the OpenCV functions. During testing, the algorithm estimated that our system had a constant velocity forwards, even though we were stationary. This error sounds like an error with the calibration procedure, although the same procedure has been used with ORB-SLAM2 and worked fine. As the source of this error was not identified, ORB-SLAM2 was instead used to extract data from our images.

ORB-SLAM2

ORB-SLAM2 is a real-time SLAM (Simultaneous Localization And Mapping) library for Monocular, Stereo and RGB-D cameras that computes the camera trajectory and a sparse 3D reconstruction [14]. In addition to solving the localization problem like in LIBVISO2, ORB-SLAM2 also solves the mapping problem: building a map of an unknown environment and localize the camera in it. ORB-SLAM2 is developed for the Ubuntu Linux OS.

ORB-SLAM2 has three main threads that run in parallel: tracking, local mapping and loop closing. The tracking thread extracts and matches feature points in the scene to estimate the cameras position and orientation in the map. The local mapping thread manages the local map and optimizes it. Finally the loop closing thread detects if the camera has returned to a scene it has previously been at, and corrects the accumulated drift.

The C++ script used to run ORB-SLAM2 with stereo images is shown in Listing B.2. This is a modification of the `stereo_kitty.cc` example script that is included with the source files. Before sending our images through the algorithm, they are rectified by the OpenCV functions. Once the algorithm is finished, the data is stored in a text file that we can import into Matlab.

3.2.4 NavLab

NavLab (Navigation Laboratory) is a generic simulation and post-processing tool for navigation, developed by the navigation group at FFI [7], [8]. NavLab is built on object oriented Matlab, which allows us to create and add new classes to expand the functionality. In this thesis it is used for post-processing of data acquired from the different sensors in our system.

As explained in section 2.6, NavLab uses an error state Kalman filter to estimate the errors in position and orientation of a system, using the difference between the IMU and external sources of data. In our case, the external data will be the position and yaw measurement from the GNSS receiver, and the delta angle measurement from the camera.

The steps in NavLab can be broken down like this: measurements → preproc → estimator → export.

Firstly we have to prepare our synchronised measurement data for input into NavLab. The preproc process accepts space-separated row vectors in the double format, gathered in a .bin file. The number of columns and what data they contain must be specified in the `sensor_pre.ini` file in NavLab. The "sensor" name is here replaced with the name of the sensor (e.g. "imu").

The preproc process is now able to read our sensor data. Preproc will split the data into .nlbin files that the estimator process reads, and also conduct any operations on the data that we might specify in the `sensor_pre.ini` file (e.g. rearrange the axes or compensate for lever arm).

The estimator now reads our sensor data and use them, and our mathematical models of the sensor errors, to estimate the systems position, attitude and velocity. The estimator integrates the measurements from the IMU using the navigation equations listed in Appendix A. When a measurement from an additional sensor is available, the error state is calculated and sent as a measurement to the Kalman filter.

When the estimator is finished the data is ready to be exported, either through NavLab's GUI, or directly in Matlab. It is also possible to run a smoothing process that uses the known future states to create a better estimate in the past.

NavLab also has a variable quality option, meaning that with every single new measurement, the sensor parameters describing that particular measurement can be specified. In order to use this, a text file named `sensor_est_quality.txt` must be added to the measurement data folder.

Note that since NavLab is copyrighted and sold commercially by Kongsberg Maritime, the code written for NavLab used in this thesis will not be listed here.

3.3 Complete System

Together with the prototype workshop at FFI, we have created a testing platform where we mounted the different sensors. This platform is a simple plank that is long enough to test several different baseline lengths between our antennas. It is also wide enough for our antennas to fit, and solid enough to withstand the weight and stress during testing. A picture of the platform with all sensors mounted can be seen in Figure 3.6.

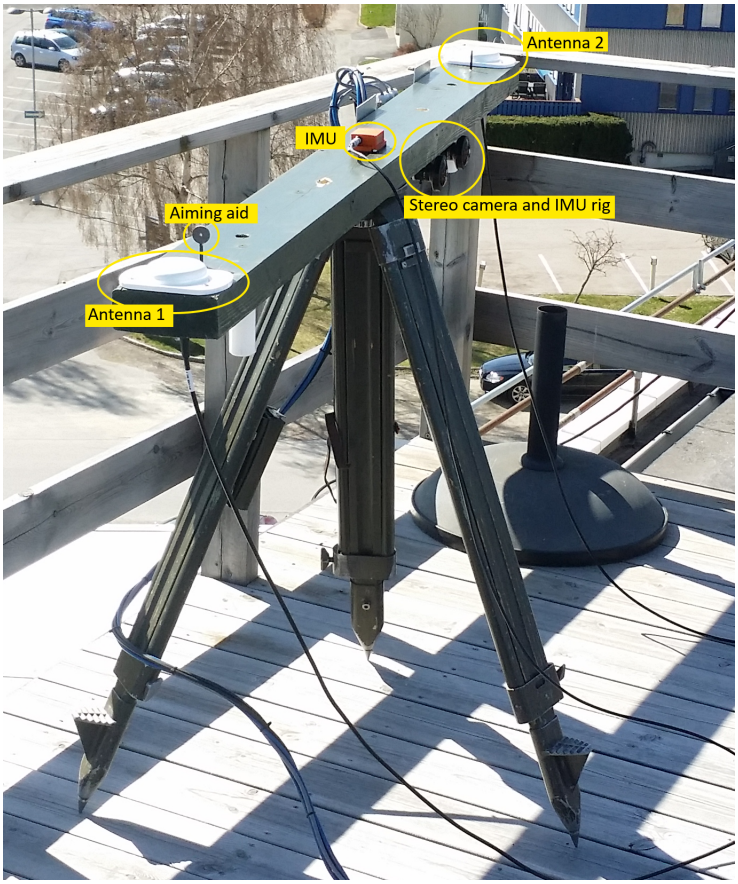

Figure 3.6: Complete test setup.

Since the antennas have their connector on the bottom, four holes have been drilled in the platform to be able to mount them. The holes are drilled such that the baseline length of the antennas will be 0.5, 1 and 1.455 meters respectively. There are four screw holes in the corners of the antennas to fasten it to the platform.

Our IMU is fastened in the centre of the platform on the top side. The IMU has three screw holes to secure it.

The stereo camera and IMU rig is mounted upside-down on the bottom forward half of the platform, as this was the simplest way to install it. This means that the cameras are facing to the right. Two aluminium plates are used to fasten the rig to the platform.

To be able to keep the platform upright and steady, threads have been put in the bottom centre

that fit the bolt of a tripod. This allows us to keep the platform fixed at a certain target, but also able to rotate it to change the heading.

In addition to all this, we have also equipped the platform with makeshift iron sights to aid in aiming. On the back of the platform is a round metallic plate with a hole in it, and on the front is a nail sharpened to a point. To aim the platform, you line up the front nail with the back hole and the object you are aiming at. An example of this can be seen in Figure 5.2.

Finally some handles have been mounted on the bottom to make the platform easier to carry when needed.

3.3.1 Coordinate Frames

Each of our sensors provide measurements represented in their own coordinate frames. We need to transform all of these measurements so they are represented in the body frame B. NavLab defines the body frame with the origin at the same position as the IMU's origin, the x-axis pointing forward, the y-axis to the right and the z-axis pointing down. Here we have used IMU 1 from the stereo camera rig, since this was the one used during the testing. Figure 3.7 shows an overview of the different coordinate frames in our system.

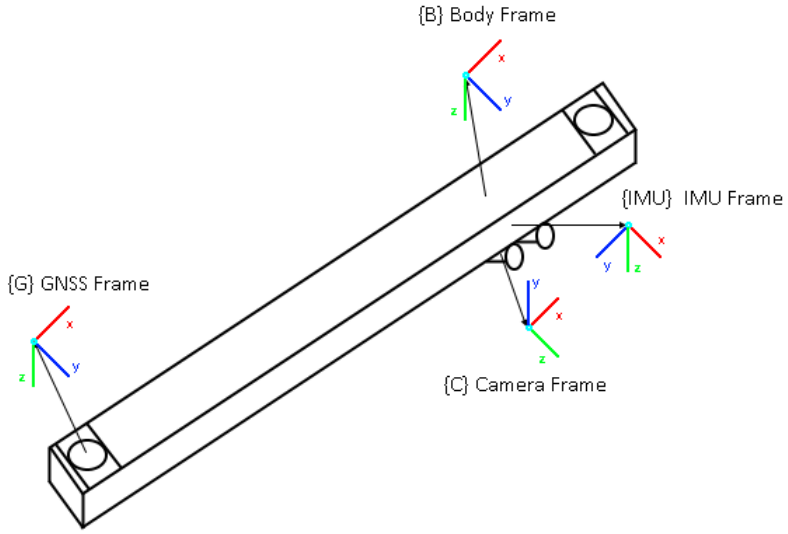

Figure 3.7: Coordinate frames in our system.

If we observe the MTi-100 IMU with the connection port facing towards us, it uses the following coordinate system: the x-axis points forwards, the y-axis points to the left, and the z-axis points upwards. As the stereo camera and IMU rig is mounted upside-down facing right, we end up with the x-axis pointing right, the y-axis pointing backwards and the z-axis pointing down relative to our platform. To transform this frame to the body frame we simply have to perform a 270° rotation along the z-axis.

The frame for the cameras have the x-axis pointing to the right in the captured image, the y-axis pointing down, and the z-axis pointing "into" the image, i.e. forwards from the camera. As again the cameras are mounted upside-down facing the right, the x-axis will point forwards, the y-axis upwards and the z-axis to the right relative to our platform. To transform this frame to the body frame we perform a 90° rotation along the x-axis.

As for the GNSS receiver, we are only interested in how the heading measurement is transformed. Since the z-axis points downwards in both the GNSS and body frame we do not need to transform this.

So far we have not mentioned the translation between the origin of the different coordinate frames. As we are interested in the heading estimation in this project, and since the IMU and camera are measuring delta-angles, and the GNSS heading vector cuts through the body frame origin, compensating for the lever arms will not have any noticeable effect on our system.

3.3.2 Mounting Errors

Since we are not able to mount all our equipment perfectly to the testing platform, there are bound to be some errors that could affect our measurements. To measure these errors we assume that the right side of our platform is perfectly straight. We also assume that our antennas are identical in size, and ignore any production errors.

Figure 3.8 shows the measurements we have taken of our testing platform after mounting all the equipment. Here, the rectangles with circles in them represents our antennas, and the red lines at each end represent our iron sight. The measurements are given in millimetres.

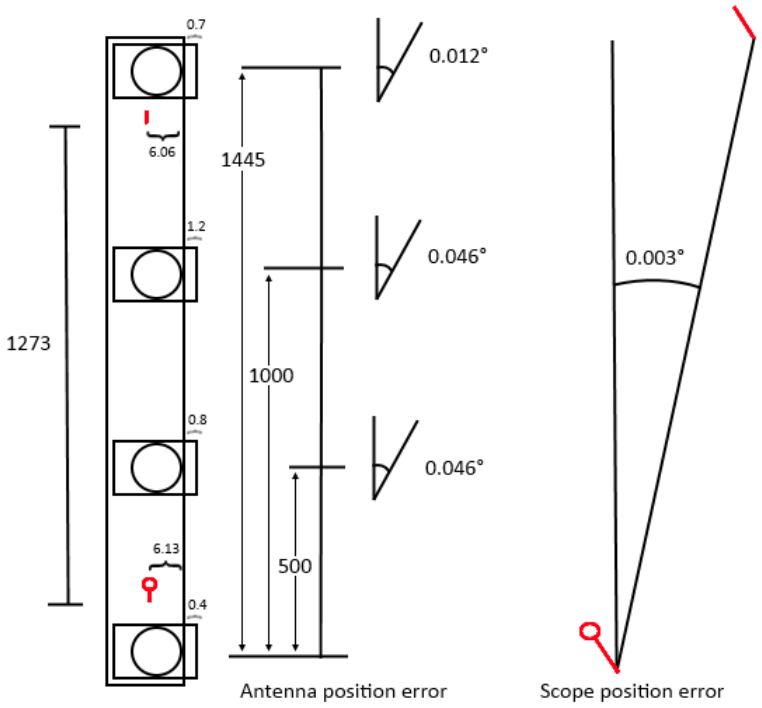

Figure 3.8: Mounting errors for our testing platform (position errors are in millimetres).

We start by looking at the error that will arise at the different baseline lengths due to the error in mounting the antennas. From Figure 3.8 we can see how far over the right side edge the antennas are sticking out, and we use this to determine the angular error we will experience.

For the longest baseline of 1.455 m, we have a difference in antenna position of 0.3 mm. The angular error therefore becomes:

$$\theta_{\text{longbaseline}} = \tan^{-1}\left(\frac{0.7 - 0.4}{1445}\right) = 0.012^\circ \tag{3.4}$$

As for the medium baseline of 1 m, the antenna position difference is 0.8 mm. The error becomes:

$$\theta_{\text{mediumbaseline}} = \tan^{-1}\left(\frac{1.2 - 0.4}{1000}\right) = 0.046^\circ \tag{3.5}$$

Finally for the short baseline of 0.5 m, the ratio of antenna position difference and baseline length stays the same. Thus the error will be equivalent:

$$\theta_{\text{smallbaseline}} = \tan^{-1}\left(\frac{0.8 - 0.4}{500}\right) = 0.046^\circ \tag{3.6}$$

We also have errors in the positioning of the iron sight we have mounted to aid in the aiming of the platform, that will affect the heading measurement. From Figure 3.8 we see that the difference in positioning between the front and rear sights is -0.07 mm. With a distance between the sights of 1.273 m, the error becomes:

$$\theta_{\text{scope}} = \tan^{-1}\left(\frac{6.06 - 6.13}{1237}\right) = -0.003^{\circ} \tag{3.7}$$

As this error is very small (aiming at a position 2 km away will give an error of 0.1 m), it will be ignored here.

Chapter 4

Sensor Models

This chapter describes the mathematical models of our sensors we have implemented in NavLab. It looks at the error models, and how we get the information in our systems on the state-space form.

4.1 IMU and GNSS

The IMU and GNSS position classes are already implemented in NavLab, so we only need to find the correct sensor parameters. As for the GNSS heading measurement, we will be using a modified compass class. The error state in all of these classes are modelled as 1. order Markov processes, as explained in Section 2.5. Our goal then becomes to find the standard deviation of the bias σ_{bias}, the standard deviation of the white measurement noise σ_ξ, and the time constant T, for each sensor.

IMU

From section 3.1.2 we see that we have most of the values we need listed from the IMU's user manual and calibration certificate. Some slight modification is needed to get them to the unit that NavLab requires.

As the parameters for the gyroscope are all listed in degrees, while NavLab requires radians, we must do a quick conversion for these:

$$\boldsymbol{\sigma}_{bias,gyro} = 0.05^\circ/s \cdot \frac{\pi}{180^\circ} = 0.000875\, rad/s \tag{4.1}$$

$$\boldsymbol{\sigma}_{\xi,gyro} = 0.015^\circ/s \cdot \frac{\pi}{180^\circ} = 0.000263\, rad/s \tag{4.2}$$

For the accelerometers, the measurement noise is given in $\mu g/\sqrt{Hz}$ while NavLab needs $m/s^{3/2}$. The bias is given in the correct unit. The parameters therefore becomes:

$$\boldsymbol{\sigma}_{bias,acc} = 0.03\, m/s^2 \tag{4.3}$$

$$\boldsymbol{\sigma}_{\xi,acc} = 80\, \mu g/\sqrt{Hz} = 80 \cdot 10^{-6} \cdot 9.81 m/s^{3/2} = 0.000785\, m/s^{3/2} \tag{4.4}$$

As for the time constant, experience working with this type of IMU at FFI suggests that $T = 600\, s$ for both the accelerometer and gyroscope measurements is a fair assumption.

GNSS

The GNSS receiver outputs the standard deviation of the actual measurements. We can therefore use these directly in NavLab, using the variable quality functionality. Before we can do this, we have to determine how much of the noise is coloured, and how much is white noise.

Experience with GNSS at FFI suggests that around 80% of the noise will be coloured, and around 20% will be white. We therefore multiply the standard deviation reported by the receiver with 0.8 and 0.2 respectively, and send these values into NavLab.

Experience also suggests that a time constant of 60 seconds is a fair assumption for the GNSS measurements.

4.2 Camera

As there is no camera class available in NavLab for our camera sensor, we have to model this completely. As previously stated, we are primarily interested in maintaining the heading of our system as accurate as possible. We will therefore investigate the angle measurements from the camera sensor. The values from camera measurements here have the subscript VO (Visual Odometry). A tilde above a value indicates a measured value, while a hat indicates a calculated value.

4.2.1 Error Model

From our visual navigation algorithm, we receive a measurement of the rotation of the camera from one frame to the next. We call this the delta rotation matrix between the body frame at time step 1, and the body frame at time step 2: $\boldsymbol{R}_{B_1B_2}$. As we saw in Section 2.3, this rotation matrix can be uniquely defined by three angles, which we gather in the vector $\boldsymbol{\theta}_{B_1B_2}$.

If we now assume that we have a high rate of measurements from the camera, and small rotations between the frames, we can approximate this vector as as the angular velocity between the body and local frame, by dividing with the time step.

$$\tilde{\boldsymbol{\omega}}^B_{LB,VO} \approx \frac{\boldsymbol{\theta}_{B_1B_2}}{\Delta t} \tag{4.5}$$

This measurement will consist of the true value $\boldsymbol{\omega}^B_{LB,VO}$, plus some error $\delta\boldsymbol{\omega}^B_{LB,VO}$ that we now have to model.

$$\tilde{\boldsymbol{\omega}}^B_{LB,VO} = \boldsymbol{\omega}^B_{LB,VO} + \delta\boldsymbol{\omega}^B_{LB,VO} \tag{4.6}$$

During testing our camera has to look at the same scene over multiple time steps to measure its movement. We can therefore assume that the measurement will have a bias $\Delta\boldsymbol{\omega}^B_{LB,VO}$, plus some white measurement noise $\boldsymbol{\xi}_{VO}$.

$$\delta\boldsymbol{\omega}^B_{LB,VO} = \Delta\boldsymbol{\omega}^B_{LB,VO} + \boldsymbol{\xi}_{VO} \tag{4.7}$$

This bias will be modelled as a 1. order Markov process driven by normal-distributed white noise:

$$\Delta\dot{\boldsymbol{\omega}}^B_{LB,VO} = -\frac{1}{\boldsymbol{T_{VO}}}\Delta\boldsymbol{\omega}^B_{LB,VO} + \boldsymbol{\gamma}_{VO} \tag{4.8}$$

Where $\boldsymbol{T_{VO}}$ is the time-constant, and:

$$-\frac{1}{\boldsymbol{T_{VO}}} = \begin{bmatrix} -\frac{1}{T_{VO,x}} & 0 & 0 \\ 0 & -\frac{1}{T_{VO,y}} & 0 \\ 0 & 0 & -\frac{1}{T_{VO,z}} \end{bmatrix} \tag{4.9}$$

The subscripts x, y and z here denotes the x-, y- and z-axis. $\boldsymbol{\gamma}_{VO}$ is the white noise which drives the bias, called the process noise, and

$$\boldsymbol{\gamma}_{VO} = \begin{bmatrix} \gamma_{VO,x} \\ \gamma_{VO,y} \\ \gamma_{VO,z} \end{bmatrix} \tag{4.10}$$

This white noise vector is normal-distributed, so we can write $\boldsymbol{\gamma}_{VO} \sim \mathcal{N}(\boldsymbol{0}, \tilde{\boldsymbol{Q}}_{VO}\delta(\tau))$, where $\tilde{\boldsymbol{Q}}_{VO}$ is the spectral density matrix. For now we assume the bias errors along the three axis are uncorrelated, and can write:

$$\tilde{\boldsymbol{Q}}_{VO} = \begin{bmatrix} \sigma^2_{\tilde{Q}_{VO},x} & 0 & 0 \\ 0 & \sigma^2_{\tilde{Q}_{VO},y} & 0 \\ 0 & 0 & \sigma^2_{\tilde{Q}_{VO},z} \end{bmatrix} \tag{4.11}$$

Future work should include investigating if there exists correlated errors. To find $\sigma^2_{\tilde{Q}_{VO},i}$, we can look at the variance of a 1. order Markov process, by inserting equation 4.8 into the predicted estimate covariance in equation 2.31:

$$\dot{\bar{\boldsymbol{P}}} = -\frac{2}{\boldsymbol{T_{VO}}}\bar{\boldsymbol{P}} + \tilde{\boldsymbol{Q}}_{\boldsymbol{VO}} \tag{4.12}$$

This equation has the solution:

$$\bar{\boldsymbol{P}}(t) = e^{-\frac{2}{\boldsymbol{T_{VO}}}(t-t_0)}\bar{\boldsymbol{P}}_{\boldsymbol{0}} + \frac{\tilde{\boldsymbol{Q}}_{\boldsymbol{VO}}\boldsymbol{T_{VO}}}{2}(1 - e^{-\frac{2}{\boldsymbol{T_{VO}}}(t-t_0)}) \tag{4.13}$$

The steady state solution ($t \to \infty$) becomes:

$$\bar{\boldsymbol{P}}_{\infty} = \frac{\tilde{\boldsymbol{Q}}_{\boldsymbol{VO}}\boldsymbol{T_{VO}}}{2} \tag{4.14}$$

We therefore get:

$$\sigma^2_{\tilde{Q}_{VO},i} = \frac{\sigma^2_{\Delta\omega_{VO,bias},i} \cdot 2}{T_{VO,i}} \tag{4.15}$$

As for the white measurement noise, we assume that is is the sum of a large number of small random components, such that it has a Gaussian distribution. We therefore assume that $\boldsymbol{\xi}_{VO,k} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{W}_{VO,k}\delta_{kj})$, where $\boldsymbol{W}_{VO,k}$ is the covariance matrix. If we again assume uncorrelated errors between our axes, this becomes:

$$\boldsymbol{W}_{VO,k} = \begin{bmatrix} \sigma^2_{\xi_{VO,k},x} & 0 & 0 \\ 0 & \sigma^2_{\xi_{VO,k},y} & 0 \\ 0 & 0 & \sigma^2_{xi_{VO,k},z} \end{bmatrix} \tag{4.16}$$

We have now completely modelled the bias and white noise errors. What is left is to determine the parameters $\sigma_{\Delta\omega_{VO,bias}}$, $\sigma_{\xi_{VO}}$ and T_{VO}.

To find the noise parameters $\sigma_{\Delta\omega_{VO,bias}}$ and $\sigma_{\xi_{VO}}$, we can look at the degree per pixel resolution in our image, and assume that our standard deviation is 1 pixel. From section 3.1.3 we found that the FOV in our images is about 98.25° in both x and y direction. With a resolution of 1200x1200 pixels in our image, this equals:

$$\sigma_{tot} = \frac{98.25^\circ}{1200p} = 0.082^\circ/p \tag{4.17}$$

We can assume that our error is slightly larger than this, due to other errors in the visual navigation algorithm like errors in the camera model, errors in 3D reconstruction, errors in motion estimation etc. We therefore assume a total standard deviation of 0.1°. After some tuning using our testing data, it seems splitting this error evenly into coloured and white noise yields the best result. Finding the time-constant is slightly trickier, but tuning again gives a time constant of 60 seconds as a good result. We therefore end up with:

$$\sigma_{\Delta\omega_{VO,bias}} = 0.05^\circ/s, \;\; \sigma_{\xi_{VO}} = 0.05^\circ/s, \;\; T_{VO} = 60\,s \tag{4.18}$$

Note that the tuning carried out to find these parameters was done on a very small data set, so the parameters might be finely tuned to these specific sets of data. For future use, a more thorough investigation should be done to see if these parameters hold for other data sets.

4.2.2 Error State

As NavLab uses an ESKF, we need to find a redundant version of our measurement to calculate our error state. As is we do not have this version, but we can make one using other known measurements and calculations:

$$\hat{\boldsymbol{\omega}}^L_{LB} = \hat{\boldsymbol{R}}_{LB}\tilde{\boldsymbol{\omega}}^B_{IB} - \hat{\boldsymbol{R}}_{LE}\hat{\boldsymbol{\omega}}^E_{IE} - \hat{\boldsymbol{\omega}}^L_{EL} \tag{4.19}$$

Here $\tilde{\boldsymbol{\omega}}^B_{IB}$ is the gyro measurement from the IMU, $\hat{\boldsymbol{\omega}}^E_{IE}$ is the known angular velocity of the earth equal to $[7.29 \cdot 10^{-5}, 0, 0]^T$, and $\hat{\boldsymbol{\omega}}^L_{EL}$ is found using the navigation equation A.2.

By multiplying $\hat{\boldsymbol{R}}_{LB}$ with our measurement $\tilde{\boldsymbol{\omega}}^B_{LB,VO}$, and subtracting this from our new redundant state, we get our error state:

$$y = \hat{\boldsymbol{\omega}}^L_{LB} - \hat{\boldsymbol{R}}_{LB}\tilde{\boldsymbol{\omega}}^B_{LB,VO} \tag{4.20}$$

4.2.3 State-Space Formulation

To add our new sensor to the ESKF in NavLab, we need to set up all the system information on the continuous-discrete state-space form, as shown in Equations 2.28 and 2.29. As our continuous process model is discretized by NavLab, we do not account for that process here.

We first look at the camera class part of the continuous process model. This is simply the 1. order Markov process we found in equation 4.8:

$$\Delta\dot{\boldsymbol{\omega}}^B_{LB,VO} = -\frac{1}{\boldsymbol{T}_{VO}}\Delta\boldsymbol{\omega}^B_{LB,VO} + \boldsymbol{\gamma}_{VO} \tag{4.21}$$

We then have the camera class part of the continuous process model in equation 2.28, and have that:

$$\boldsymbol{F} = -\frac{1}{\boldsymbol{T}_{VO}},\ \boldsymbol{L} = 0,\ \boldsymbol{G} = 1 \tag{4.22}$$

and

$$\boldsymbol{x} = \Delta\boldsymbol{\omega}^B_{LB,VO},\ \boldsymbol{u} = 0,\ \boldsymbol{\gamma} = \boldsymbol{\gamma}_{VO} \tag{4.23}$$

The spectral density matrix of $\boldsymbol{\gamma}$ is given in equation 4.11.

We will now find the discrete measurement model. Our base is the error state we found in equation 4.20. By replacing the measured and calculated values with the true value plus the error, we get:

$$\begin{aligned} y = (\boldsymbol{R}_{LB} + \delta\boldsymbol{R}_{LB})(\boldsymbol{\omega}^B_{IB} + \delta\boldsymbol{\omega}^B_{IB}) - (\boldsymbol{R}_{LE} + \delta\boldsymbol{R}_{LE})(\boldsymbol{\omega}^E_{IE} + \delta\boldsymbol{\omega}^E_{IE}) - (\boldsymbol{\omega}^L_{EL} + \delta\boldsymbol{\omega}^L_{EL}) \\ -(\boldsymbol{R}_{LB} + \delta\boldsymbol{R}_{LB})(\boldsymbol{\omega}^B_{LB} + \Delta\boldsymbol{\omega}^B_{LB,VO} + \xi_{VO}) \end{aligned} \tag{4.24}$$

First order approximation gives:

$$\begin{aligned} y = \boldsymbol{R}_{LB}\boldsymbol{\omega}^B_{IB} - \boldsymbol{R}_{LE}\boldsymbol{\omega}^E_{IE} - \boldsymbol{\omega}^L_{EL} - \boldsymbol{R}_{LB}\boldsymbol{\omega}^B_{LB} + \delta\boldsymbol{R}_{LB}\boldsymbol{\omega}^B_{IB} + \boldsymbol{R}_{LB}\delta\boldsymbol{\omega}^B_{IB} - \delta\boldsymbol{R}_{LE}\boldsymbol{\omega}^E_{IE} \\ -\boldsymbol{R}_{LE}\delta\boldsymbol{\omega}^E_{IE} - \delta\boldsymbol{\omega}^L_{EL} - \delta\boldsymbol{R}_{LB}\boldsymbol{\omega}^B_{LB} - \boldsymbol{R}_{LB}\Delta\boldsymbol{\omega}^B_{LB,VO} - \boldsymbol{R}_{LB}\xi_{VO} \end{aligned} \tag{4.25}$$

We can now remove the terms without errors in them since these are equal:

$$\begin{aligned} y = \delta\boldsymbol{R}_{LB}\boldsymbol{\omega}^B_{IB} + \boldsymbol{R}_{LB}\delta\boldsymbol{\omega}^B_{IB} - \delta\boldsymbol{R}_{LE}\boldsymbol{\omega}^E_{IE} - \boldsymbol{R}_{LE}\delta\boldsymbol{\omega}^E_{IE} - \delta\boldsymbol{\omega}^L_{EL} \\ -\delta\boldsymbol{R}_{LB}\boldsymbol{\omega}^B_{LB} - \boldsymbol{R}_{LB}\Delta\boldsymbol{\omega}^B_{LB,VO} - \boldsymbol{R}_{LB}\xi_{VO} \end{aligned} \tag{4.26}$$

By using 2.16 we get:

$$\begin{aligned} y = S(\boldsymbol{e}^L_{LB})\boldsymbol{R}_{LB}\boldsymbol{\omega}^B_{IB} + \boldsymbol{R}_{LB}\delta\boldsymbol{\omega}^B_{IB} - S(\boldsymbol{e}^L_{LE})\boldsymbol{R}_{LE}\boldsymbol{\omega}^E_{IE} - \boldsymbol{R}_{LE}\delta\boldsymbol{\omega}^E_{IE} - \delta\boldsymbol{\omega}^L_{EL} \\ -S(\boldsymbol{e}^L_{LB})\boldsymbol{R}_{LB}\boldsymbol{\omega}^B_{LB} - \boldsymbol{R}_{LB}\Delta\boldsymbol{\omega}^B_{LB,VO} - \boldsymbol{R}_{LB}\xi_{VO} \end{aligned} \tag{4.27}$$

Or:

$$\begin{aligned} y = S(\boldsymbol{e}^L_{LB})\boldsymbol{\omega}^L_{IB} + \boldsymbol{R}_{LB}\delta\boldsymbol{\omega}^B_{IB} - S(\boldsymbol{e}^L_{LE})\boldsymbol{\omega}^L_{IE} - \delta\boldsymbol{\omega}^L_{IE} - \delta\boldsymbol{\omega}^L_{EL} \\ -S(\boldsymbol{e}^L_{LB})\boldsymbol{\omega}^L_{LB} - \boldsymbol{R}_{LB}\Delta\boldsymbol{\omega}^B_{LB,VO} - \boldsymbol{R}_{LB}\xi_{VO} \end{aligned} \tag{4.28}$$

By using A.6 and A.8 from the error propagation of the navigation equations in appendix A, we get:

$$y = S(\boldsymbol{e}^L_{LB})\boldsymbol{\omega}^L_{IB} + \boldsymbol{R}_{LB}\delta\boldsymbol{\omega}^B_{IB} - S(\boldsymbol{e}^L_{LE})\boldsymbol{\omega}^L_{IE} + S(\boldsymbol{e}^L_{EL})\boldsymbol{\omega}^L_{IE} - \frac{1}{\boldsymbol{r}_{EB}}S(\boldsymbol{u}^L_{EB})\delta\boldsymbol{v}^L_{EB} \\ -S(\boldsymbol{e}^L_{LB})\boldsymbol{\omega}^L_{LB} - \boldsymbol{R}_{LB}\Delta\boldsymbol{\omega}^B_{LB,VO} - \boldsymbol{R}_{LB}\xi_{VO} \tag{4.29}$$

Or:

$$y = S(\boldsymbol{e}^L_{LB})(\boldsymbol{\omega}^L_{IB} - \boldsymbol{\omega}^L_{LB}) + \boldsymbol{R}_{LB}\delta\boldsymbol{\omega}^B_{IB} + 2S(\boldsymbol{e}^L_{EL})\boldsymbol{\omega}^L_{IE} - \frac{1}{\boldsymbol{r}_{EB}}S(\boldsymbol{u}^L_{EB})\delta\boldsymbol{v}^L_{EB} - \\ \boldsymbol{R}_{LB}\Delta\boldsymbol{\omega}^B_{LB,VO} - \boldsymbol{R}_{LB}\xi_{VO} \tag{4.30}$$

We can also make the following simplification for $\boldsymbol{\omega}^L_{IB} - \boldsymbol{\omega}^L_{LB}$:

$$\boldsymbol{\omega}^L_{IB} - \boldsymbol{\omega}^L_{LB} = \boldsymbol{R}_{LB}\boldsymbol{\omega}^B_{IB} - \boldsymbol{R}_{LB}\boldsymbol{\omega}^B_{IB} + \boldsymbol{R}_{LE}\boldsymbol{\omega}^E_{IE} + \boldsymbol{\omega}^L_{EL} = \boldsymbol{\omega}^L_{IE} + \boldsymbol{\omega}^L_{EL} \tag{4.31}$$

We then finally end up with:

$$y = \begin{bmatrix} -S(\boldsymbol{\omega}^L_{IE} + \boldsymbol{\omega}^L_{EL}) & -\frac{1}{\boldsymbol{r}_{EB}}S(\boldsymbol{u}^L_{EB}) & -2S(\boldsymbol{\omega}^L_{IE}) & \boldsymbol{R}_{LB} & 0 & -\boldsymbol{R}_{LB} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} -\boldsymbol{R}_{LB}\xi_{VO} \end{bmatrix} \tag{4.32}$$

Here, the state vector $\boldsymbol{x}$ have been augmented with the error states from the navigation equations and the IMU class, and is equal to:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{e}^L_{LB} \\ \delta\boldsymbol{v}^L_{EB} \\ \boldsymbol{e}^L_{EL} \\ \Delta\boldsymbol{\omega}^B_{IB} \\ \Delta\boldsymbol{f}^B_{IB} \\ \Delta\boldsymbol{\omega}^B_{LB,VO} \end{bmatrix} \tag{4.33}$$

We then have the camera class part of the discrete measurement model in Equation 2.29, and have that:

$$\boldsymbol{H}_k = \begin{bmatrix} -S(\boldsymbol{\omega}^L_{IE,k} + \boldsymbol{\omega}^L_{EL,k}) & -\frac{1}{\boldsymbol{r}_{EB,k}}S(\boldsymbol{u}^L_{EB,k}) & -2S(\boldsymbol{\omega}^L_{IE,k}) & \boldsymbol{R}_{LB,k} & 0 & -\boldsymbol{R}_{LB,k} \end{bmatrix}, \\ \boldsymbol{\xi}_k = -\boldsymbol{R}_{LB,k}\xi_{VO,k} \tag{4.34}$$

The covariance matrix for $\boldsymbol{\xi}_k$ is now altered from Equation 4.16, since we are multiplying with $\boldsymbol{R}_{LB}$:

$$\boldsymbol{W}_k = cov(\boldsymbol{\xi}_k) = \boldsymbol{R}_{LB,k}\xi_{VO,k}(\boldsymbol{R}_{LB,k}\xi_{VO,k})^T = \boldsymbol{R}_{LB,k}\boldsymbol{W}_{VO,k}\boldsymbol{R}_{BL,k} \tag{4.35}$$

We now have the complete system model on the state space form, and can write this into a camera class in NavLab, that allows us to add measurements from the camera to the ESKF.

Chapter 5

Results

This chapter describes the different tests performed with our platform, and shows the results from these tests. In total, three types of tests were performed. In the first test we addressed how well the GNSS receiver performs with different antenna baselines. The second test is to find how the GNSS receiver responds to being jammed. Finally, in the last test we run the complete system to see how well our camera can aid the IMU in estimating the heading when the GNSS signal is not available.

All our tests were performed on the roof of FFI. We aimed our platform at Røverkollen telecentre, which is a 84 meter high telecommunication tower located north-east of Oslo. As we can see in Figure 5.1, the tower is located 8424.27 meters from our testing point, with an angle of 273.58° relative to north. This angle was also confirmed using the azimuth function in Matlab's Mapping Toolbox.

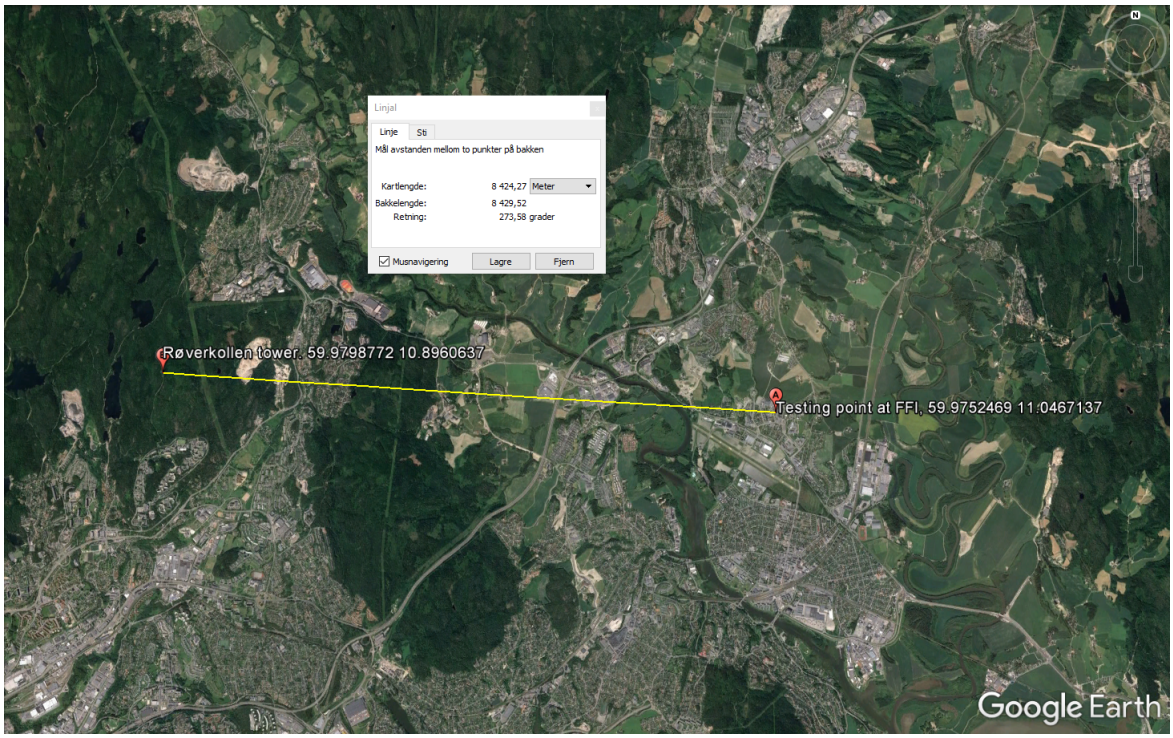

Figure 5.1: Map of testing area.

Figure 5.2 shows the testing platform aiming at the tower. The tower might be difficult to see in the picture due to foliage, but it provided a clear point for us to aim at during the tests.

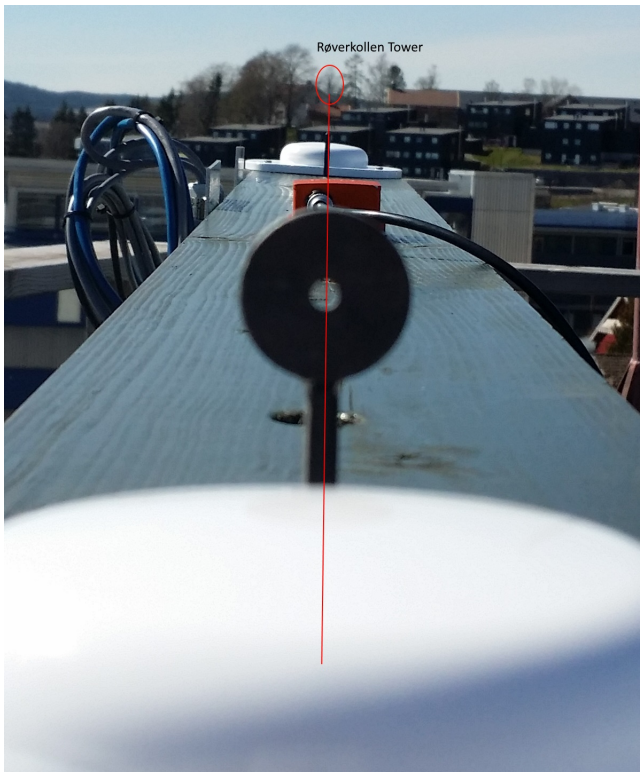

Figure 5.2: Aiming at Røverkollen telecentre.

5.1 GNSS Receiver Baseline Test

During this test, our goal is to see how accurate the GNSS receiver can measure heading, with different antenna baseline lengths. As previously mentioned, the practical maximum baseline length we get between our antennas on a standard assault rifle is about 0.5 meters. If our tests indicate that this baseline does not provide sufficient accuracy, an idea is to mount some sort of extension to the rifle that will give us a longer baseline. For this reason we have also tested the GNSS with a few longer baselines. As we saw in Section 3.3, we have made mounting holes for our antennas at four locations on the platform, giving us the following possible baseline lengths:

1. The “long” baseline is 1.445 meters long
2. The “medium” baseline is 1 meter long
3. The “small” baseline is 0.5 meters long

For each baseline length we completed 9 tests, each of 1 minute duration, logging heading with a frequency of 1 Hz, and powering on/off the GNSS receiver between each 1 minute test. During all tests the platform was directed at Røverkollen telecentre. The following sections shows the results from these tests.

Long baseline

In the long baseline test we set up our antennas at the ends of the testing platform, giving us a total baseline of 1.455 meters. Figure 5.3 shows the results from the first of the 9 tests done at this baseline length, while the rest of the tests are summed in Table 5.1. This table shows the mean heading for each test, and its standard deviation. It also shows the error between the mean heading and the true heading. The table also includes one run were we have calculated the total mean and standard deviation using all the samples in all the runs.

From Section 3.3.2 we saw that our error in mounting the antennas will cause an angular error of 0.012° for this baseline length. The true heading for this test is therefore 273.59°.

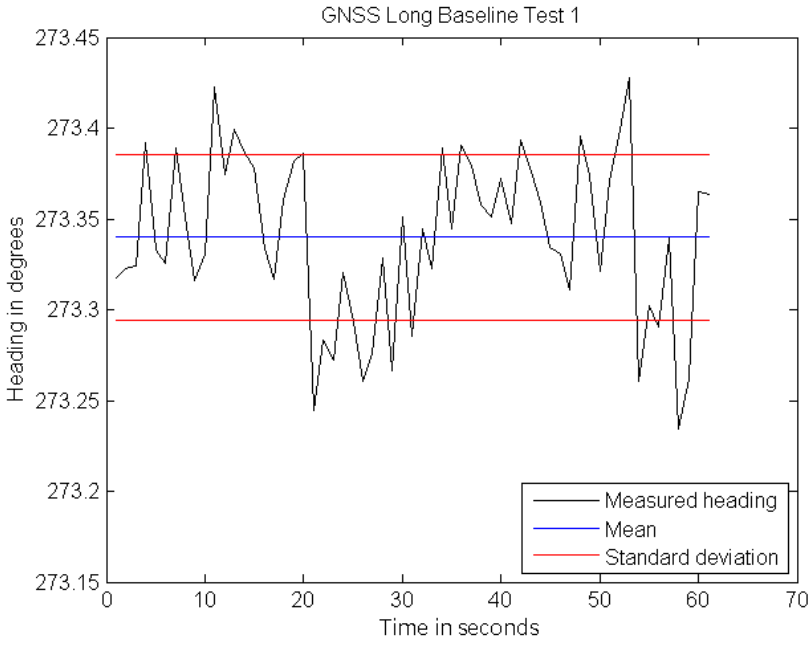

Figure 5.3: Long baseline test 1, showing the heading measurement series, with its mean and standard deviation.

Run	Mean heading	Standard deviation	Error from true heading
1.	273.3399	0.0457	-0.2501
2.	273.2358	0.0354	-0.3522
3.	273.2998	0.0457	-0.2902
4.	273.4188	0.0909	-0.1712
5.	273.4519	0.0818	-0.1381
6.	273.3052	0.0689	-0.2848
7.	273.4141	0.0640	-0.1759
8.	273.3897	0.0432	-0.2003
9.	273.2711	0.0672	-0.3189
All samples	273.3297	0.1049	-0.3003

Table 5.1: Results from long baseline test, showing the mean and standard deviation of each test, and the difference from the mean to the true heading.

Medium Baseline

In the medium baseline test we move the front antenna to the second to last mounting spot, leaving the reference antenna at the same position, and giving us a total baseline of 1 meter. Figure 5.4 shows the results from the first of the 9 tests done at this baseline length, while the rest of the tests are summed up in Table 5.2.

Section 3.3.2 shows that our error in mounting the antennas will cause an angular error of 0.046° for this baseline length. The true heading for this test is therefore 273.63°.

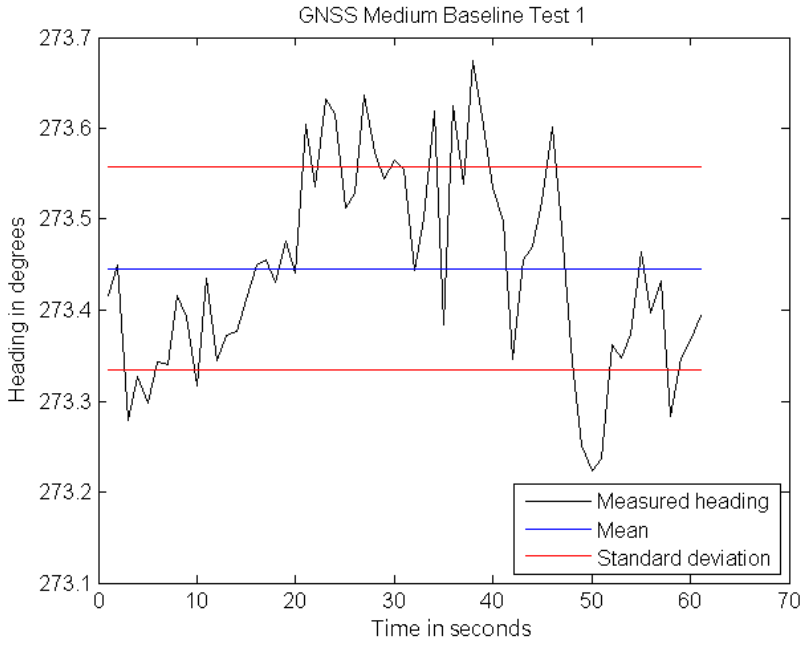

Figure 5.4: Medium baseline test 1, showing the heading measurement series, with its mean and standard deviation.

Run	Mean heading	Standard deviation	Error from true heading
1.	273.4266	0.1270	-0.2034
2.	273.3804	0.0907	-0.2496
3.	273.5237	0.1413	-0.1063
4.	273.6558	0.1382	0.0258
5.	273.3412	0.1058	-0.2888
6.	273.2285	0.0999	-0.4015
7.	273.4961	0.1297	-0.1339
8.	273.2977	0.1152	-0.3323
9.	273.3303	0.1032	-0.2997
All samples	273.3747	0.1951	-0.2553

Table 5.2: Results from medium baseline test, showing the mean and standard deviation of each test, and the difference from the mean to the true heading.

Small Baseline

In the small baseline test we move the front antenna again to the position closest to the back antenna, giving us a baseline of 0.5 meters. Figure 5.5 shows the results from the first of the 9 tests done at this baseline length, while the rest of the tests are summed up in Table 5.3.

From section 3.3.2 we saw that the angular error caused by the error in mounting the antennas remained the same as for the medium baseline. The true heading for this test is therefore the same as from the medium baseline test, 273.63°.

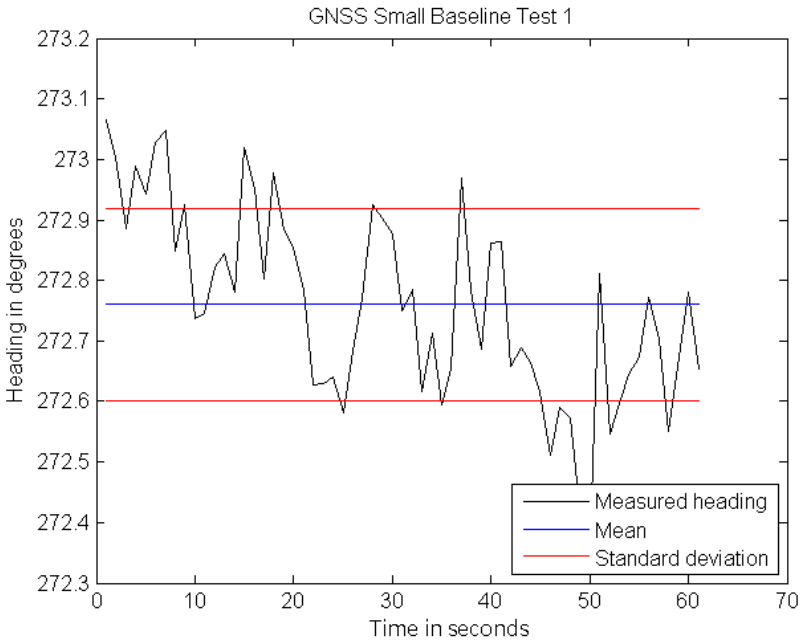

Figure 5.5: Small baseline test 1, showing the heading measurement series, with its mean and standard deviation.

Run	Mean heading	Standard deviation	Error from true heading
1.	272.7999	0.2144	-0.8301
2.	273.0067	0.1572	-0.6233
3.	273.0572	0.1653	-0.5728
4.	273.0331	0.4312	-0.5969
5.	273.1841	0.1502	-0.4459
6.	272.9076	0.2111	-0.7224
7.	273.0679	0.1845	-0.5621
8.	272.8558	0.2102	-0.7742
9.	273.0425	0.1884	-0.5875
All samples	272.9381	0.2533	-0.6919

Table 5.3: Results from small baseline test, showing the mean and standard deviation of each test, and the difference from the mean to the true heading.

5.2 GNSS Receiver Jamming Test

In this test we tried to jam the GNSS receiver on our platform to see how it reacted. This was done by using a very low power GPS-jammer on the L1-frequency (1575.42 MHz), with a 20 MHz bandwidth.

Two tests were conducted, one with the short baseline between antennas, see Figure 5.6, and one with the long baseline between antennas, see Figure 5.7.

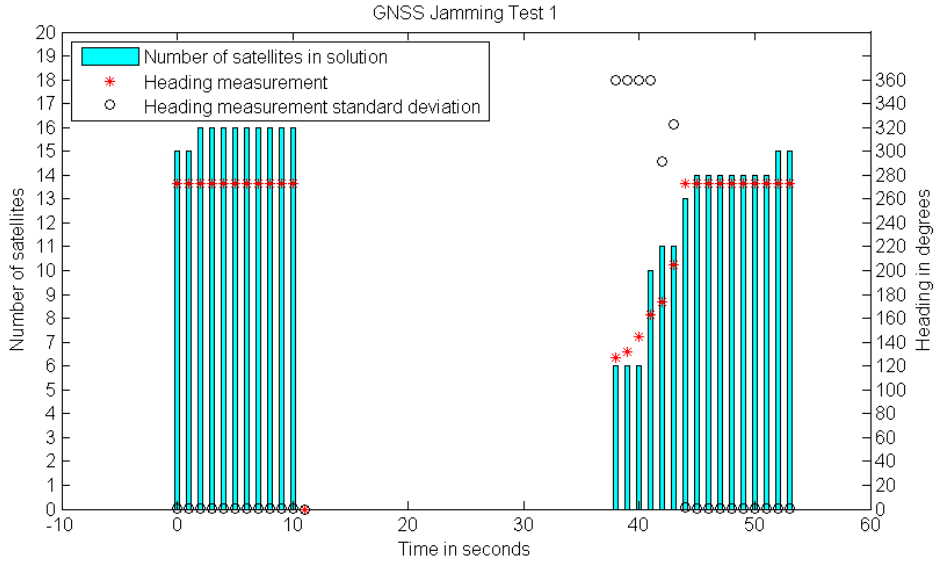

Figure 5.6: Jamming test 1, showing the estimated heading and its standard deviation from the GNSS receiver, and the amount of satellites used in the solution. The jamming starts at 10 seconds. The small baseline between the antennas is used.

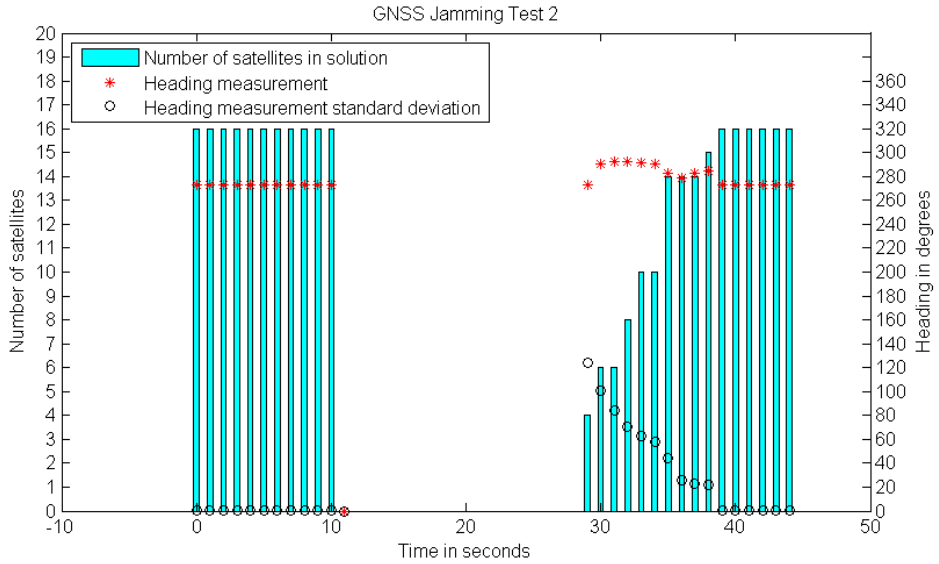

Figure 5.7: Jamming test 2, showing the estimated heading and its standard deviation from the GNSS receiver, and the amount of satellites used in the solution. The jamming starts at 10 seconds. The long baseline between the antennas is used.

As seen in both Figure 5.6 and 5.7, as soon as the jamming is started, the heading measurement immediately drops to zero. This could pose a problem if we only look at the measurement itself, but as we can see, the number of satellites used in the solution also drops to zero. The solution status

parameter in the log is also changed from `SOL_COMPUTED` to `INSUFFICIENT_OBS`. If we monitor these parameters, and discard any measurement with the incorrect status, jamming should not cause us to receive faulty measurements.

As the heading log only updates on change, no measurements are received until the jamming is stopped. Once resumed, we see that the GNSS receiver needs a few seconds to acquire enough satellites to get a correct heading measurement. We can see how the standard deviation of the heading measurement given by the GNSS receiver reflect how many satellites it uses in the solution.

5.3 Visual Navigation Test

The tests of the visual navigation uses our IMU and the stereo camera rig to see how well we can keep our heading when we lose the GNSS signal. In total four tests were completed, with a varied amount of movement of the platform in each one. The initial heading is first established using measurements from the GNSS receiver, but are then disabled once the camera data starts. The estimated heading is presented for each test both when the IMU is acting alone, and when the camera is assisting in the estimation.

Unfortunately, because of the long setup time and the limited availability of the stereo cameras, no further tests were completed. The small amount of test data we acquired does provide some problems when we try to extract any statistical information from them, as we will see in Chapter 6.

Since the extraction of the images takes a very long time, these tests were run with the camera providing data at the rate of 10 Hz. One of the tests has also been run at 70 Hz to check if this has any effect on the result.

During analysis of the data, it was discovered that the IMU causes so much drift in position that it has an adverse effect on the estimated orientation. Because of this, the GNSS position data has been used through the entire test, even when we assume we lost the GNSS signal. We did this to get a more realistic result from our tests, since in the final product the camera would also be used to estimate the translational motion of the system, thereby reducing the translational drift from the IMU.

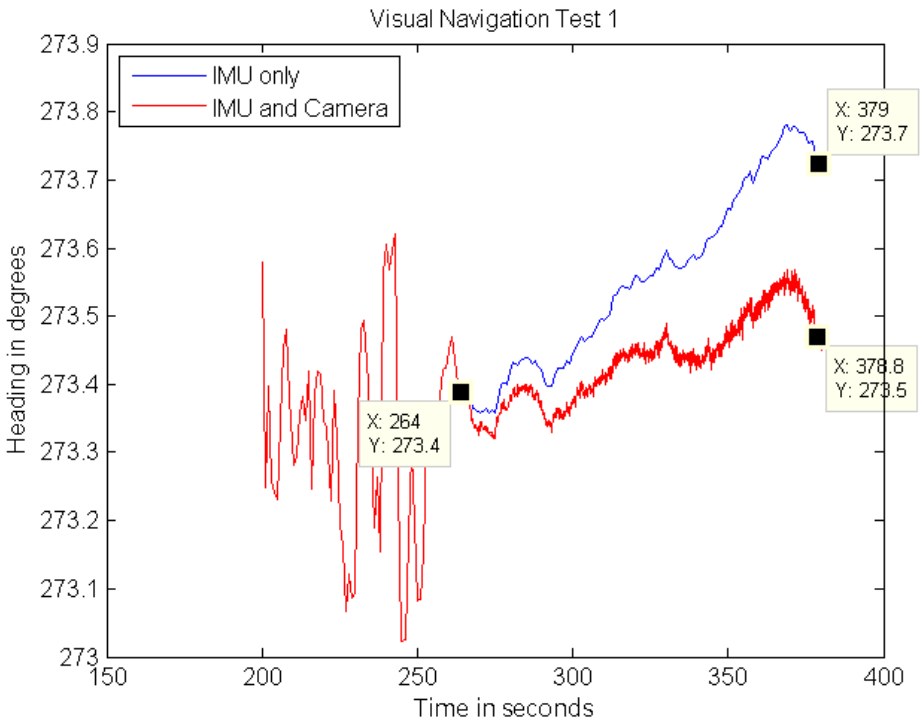

Figure 5.8: Visual navigation test 1, showing the estimated heading with both the IMU alone, and the IMU + camera providing data. The period where the camera data is available is from $t = 264.3$ to $t = 378.8$.

Test 1

This test was done while the platform was standing completely still. The data in this test was mainly taken during the setup phase, so the cameras did not start recording until quite late in the test. Figure 5.8 shows the estimated heading for the case when only the IMU data is active, and the case when both the IMU and camera are providing data. As seen in the figure, the estimated heading at the start and stop points for the camera data are highlighted. The period where the camera data is available is from $t = 264.3$ to $t = 378.8$. Until this point we use the GNSS heading data, which is why the estimated heading is so erratic.

From Figure 5.8 we can see that with only the IMU estimating the heading, a heading error of 0.3° is accumulated over 114.5 seconds. When we include the camera data in the estimation, this error drops to 0.1°.

Test 2

This test was completed with the platform first standing still, then slowly rotated back and forth around the z-axis to create disturbances in the heading. The platform was then re-positioned to its original heading and left stationary until the end. Figure 5.9 shows the estimated heading with both the IMU acting alone, and the camera assisting in the heading estimation. The period where the camera data is available is from $t = 25.7$ to $t = 133.9$.

During this test, the feet of the tripod our platform is using were not completely secure, causing the entire platform to drop slightly in the back. This made it difficult to aim it correctly after its movement, so the final heading could be slightly wrong because of this.

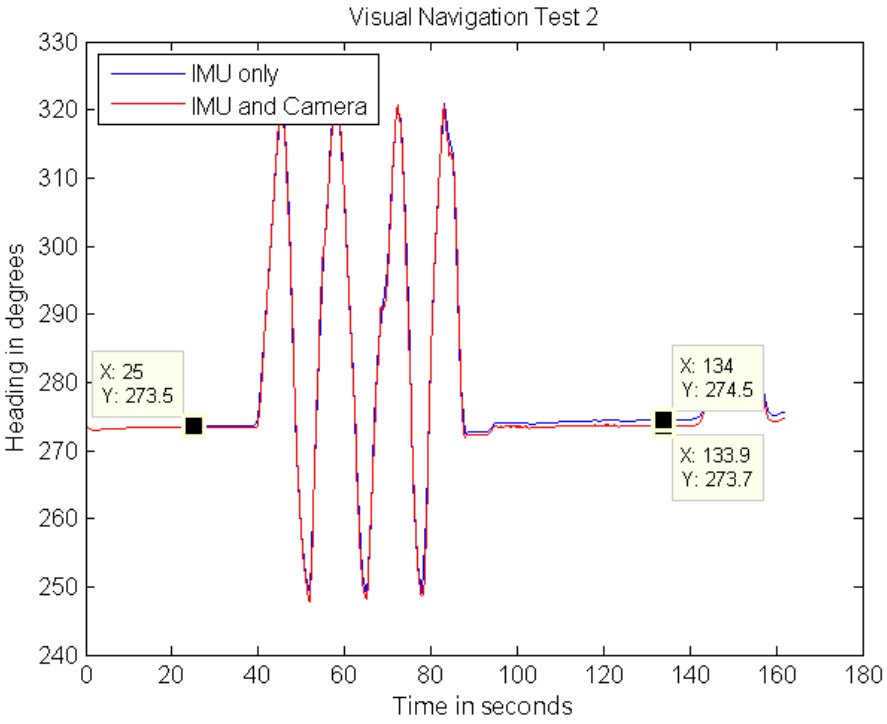

Figure 5.9: Visual navigation test 2, showing the estimated heading with both the IMU alone, and the IMU + camera providing data. The period where the camera data is available is from $t = 25.7$ to $t = 133.9$.

Figure 5.9 shows an accumulated heading error of 1.0° over 108.2 seconds when only the IMU is estimating heading. Including the camera data into this estimation improves the error to 0.2°.

Test 3

Since we experienced some problems with re-aiming our platform at the end of test 2, another test was taken with the same slow rotation in yaw. Figure 5.10 shows the resulting heading when both

the IMU is acting alone, and when the camera is assisting in the estimation. The period where the camera data is available is from $t = 11.9$ to $t = 140.7$.

In this test we also extracted the images from the camera at 70Hz, to see if this would have any effect on the estimation.

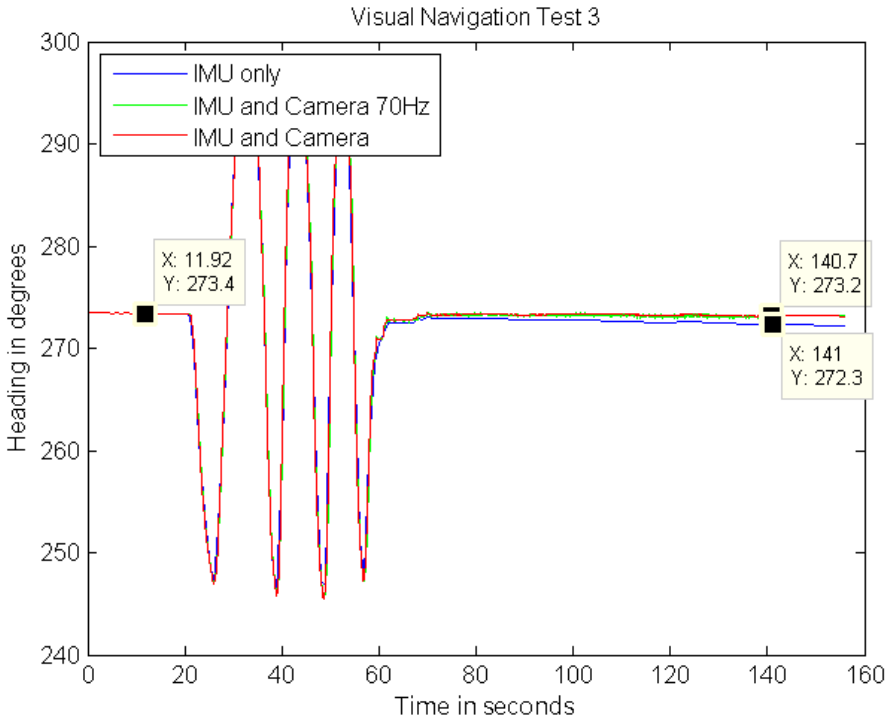

Figure 5.10: Visual navigation test 3, showing the estimated heading with both the IMU alone, and the IMU + camera providing data. The period where the camera data is available is from $t = 11.9$ to $t = 140.7$.

From Figure 5.10 we can see that with only the IMU estimating the heading, a heading error of 1.1° is accumulated over 128.8 seconds. When we include the camera data in the estimation, this error drops to 0.2°. In addition, Figure 5.10 shows us that there is no improvement in the heading estimate when running the camera data at 70Hz. This could be because of our slow rotation of the platform, allowing the camera to easily detect several feature points over multiple time steps. In hindsight, we should have run the camera data at 70Hz in test 4, were we have a much faster yaw rotation.

Test 4

In the final test we carried out the same movements as in test 2 and 3, but this time at a much higher speed. Figure 5.11 shows the estimated heading for the case when only the IMU data is active, and for the case when both the IMU and camera are providing data. The period where the camera data is available is from $t = 13.6$ to $t = 99.6$.

Figure 5.11 shows an accumulated heading error of 0.6° over 86.0 seconds when only the IMU is estimating heading. Including the camera data into the estimation improves the error to 0.3°.

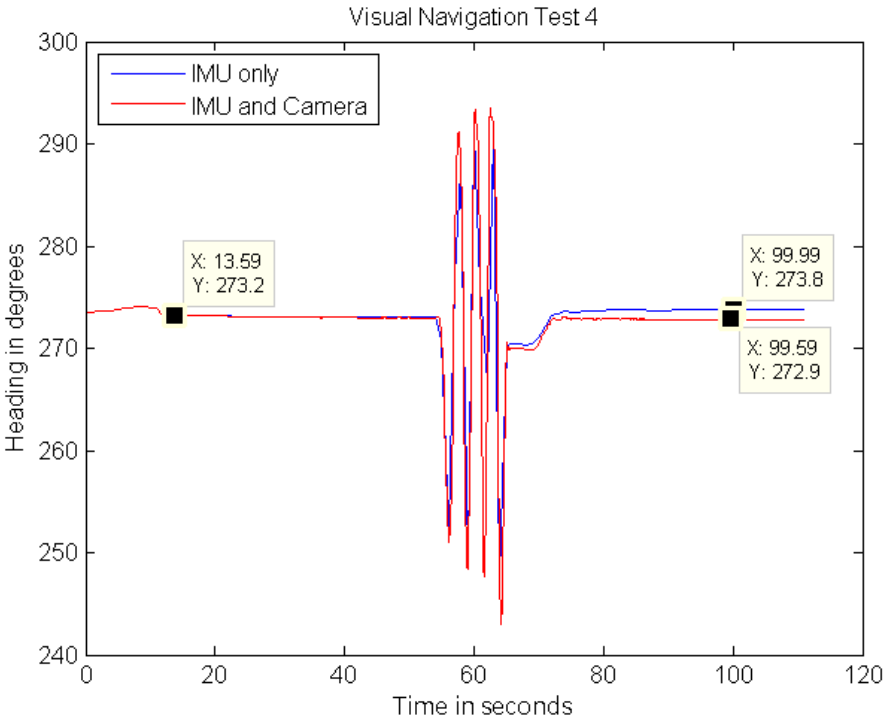

Figure 5.11: Visual navigation test 4, showing the estimated heading with both the IMU alone, and the IMU + camera providing data. The period where the camera data is available is from $t = 13.6$ to $t = 99.6$.

Chapter 6

Discussion

In this chapter the results from the different tests are discussed, and suggestions to further work is provided.

6.1 Dual Antenna GNSS Performance

The main goal for the dual antenna GNSS test was to see how the heading estimate was related to different baseline lengths. This was because specifications for the GNSS receiver only state heading accuracy with a baseline of 2.0 meters, while the maximum length on a standard assault rifle is about 0.5 meters. The receiver was therefore tested with a baseline length of 0.5 meters, but also with baseline lengths of 1.445 and 1.0 meters, in case the smallest baseline did not provide a sufficient estimate. The specifications of the GNSS receiver are also presumed to be based on ideal conditions, while in the real world we may for example have reflecting surfaces that create multipath disturbances.

We would assume that the shorter baseline we test with, the lower the accuracy and precision will get. Accuracy is a measure of how far the mean of a set of samples is from the true value, and precision is a measure of the distribution of those samples. In Figure 6.1 we have fitted a normal distribution to the total set of samples for each baseline length, to illustrate the accuracy and precision results from the different tests.

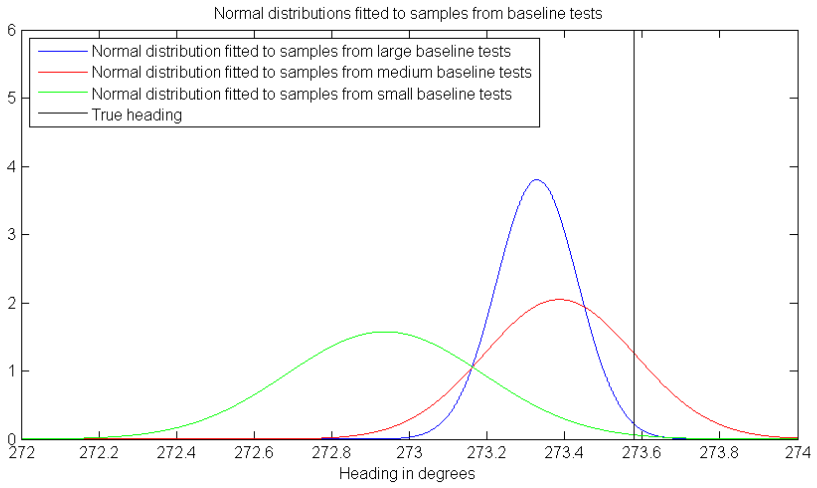

Figure 6.1: Normal distributions fitted to the total amount of samples from each baseline test, to illustrate the difference in precision and accuracy.

As we can see from Figure 6.1, the precision of the samples drop as we lower the baseline length, as assumed. However we can see that the accuracy of the medium baseline samples are better than the large one. This result could be explained by slight fluctuations in our aiming accuracy, or changes in the satellite constellation (e.g. satellites no longer tracked/new satellites in view). It

should also be mentioned that we have calculated the accuracy based on only one measurement series for each baseline, which is too little information for us to draw precise conclusions. More test series should be conducted to determine this relationship.

The small baseline is the most interesting to look at, since this is about the baseline length we can expect on a standard assault rifle. Even though the result is worse than from the medium and large baseline, the error for the small baseline stays well within 1°. We will look at how this error affects the total target localization error later in this chapter.

6.2 Heading Estimation With IMU and Camera

Since the reliability of GNSS signals on the modern battlefield are questionable because of the ease of jamming these, our system has redundant sensors in place in the form of an IMU and camera to maintain the accuracy of the estimated heading when GNSS is not available. The IMU and camera data has been fused using NavLab to create a heading estimate. The four tests we completed, described in Section 5.3, are summed up in Table 6.1, showing the heading drift with and without camera data, and the time period from the beginning to the end of the run. The table also shows the movement in yaw carried out during the test.

Test number	Only IMU	IMU and camera	Time	Movement in yaw
1	0.3°	0.1°	114.5 s	None
2	1.0°	0.2°	108.2 s	Slow
3	-1.1°	-0.2°	128.8 s	Slow
4	0.6°	-0.3°	86.0 s	Fast

Table 6.1: Results from the visual navigation tests, showing drift in heading estimate with and without camera data.

6.2.1 Only IMU

The second column in Table 6.1 shows heading drift of the IMU. This drift in yaw happens due to integration of biased and noisy gyroscope data, compared to the roll and pitch that in a static situation is calculated using the gravitational vector directly. As we can see, heading drift is about ±1° in time period of about 2 minutes. This drift is quite significant, and will cause a position error of ±35 meters at a range of 2000 meters. If the drift keeps increasing at this rate, the estimated heading will soon be completely useless for our cause. It is therefore clear that we need an additional sensor to support the IMU and reduce this drift. For this reason we have added visual navigation data from our cameras.

6.2.2 Adding Camera Data

The third column in Table 6.1 shows the drift in heading when combining the IMU and camera data. As we can see, the drift is now significantly reduced for all of our tests, with around ±0.2° drift in a time period of about 2 minutes. The drift is still present due to errors in the visual navigation algorithm, e.g. the pixel resolution error. If we compare our result now from when we only used the IMU data, our new drift of ±0.2° will cause a position error of only ±7 meters at a range of 2000 meters.

Table 6.1 also shows that test 1 had no movement of the platform, test 2 and 3 had slow movement in yaw, and test 4 had rapid movement in yaw. It seems then from looking at the resulting drift in Table 6.1 that the more rapid movement our system experiences, the worse the camera is able to compensate for the drift. It is difficult to draw any firm conclusions from the test results, since we have a limited amount of samples available.

In this estimation we have not taken advantage of the loop closing feature of ORB-SLAM2, i.e. that it can correct for drift when the camera returns to a scene it has seen before. An idea could be to have a method of switching between this feature when the soldier is stationary, and turning it off when he is in motion to save processing power and storage space. This would reduce the total accumulated drift, since the estimation would only drift when the soldier is in motion. This would only work with SLAM algorithms though, and not pure visual odometry algorithms.

6.3 Total Target Localization Error

One of the system requirements is that the total target localization error should not exceed 30 m circular error probable (CEP). CEP is a measure of the spread in the horizontal plane, and is equal to a circle with radius r_{50} one will hit within with 50% probability. The relationship between the CEP and the standard deviation is given in [23]:

$$CEP = r_{50} = \sqrt{-2\ln(1-1/2)}\,\sigma \tag{6.1}$$

The CEP assumes, as the name suggests, that the error in the horizontal plane is circular. This means that the uncertainty in side and length must be equal, $\sigma_{x,tot} = \sigma_{y,tot}$. If the uncertainty in side and length is approximately equal, $\sigma_{x,tot} \approx \sigma_{y,tot}$, we can find an approximate standard deviation that is equal in side and length [23]:

$$\sigma^2 = \frac{1}{2}(\sigma^2_{x,tot} + \sigma^2_{y,tot}) \tag{6.2}$$

For us this will be the case only at one point, since our heading error will cause a side error that will change depending on the distance to the target, while the error in length, measured by a laser rangefinder, will stay constant. Since our requirement is using CEP, we will approximate a circular error using Equation 6.2, although this is not ideal since our error in general will be elliptical.

We will be looking at the total target localization error in the case were we have the smallest baseline, since this is the most realistic baseline length. From section 5.1 we found that all our measurements over all our tests with the smallest baseline had a standard deviation σ_θ of 0.2533°, with an error from true heading, or bias, of -0.6919°. This bias seems quite consistent for all our tests, so we can assume that this is an error we can remove with calibration (e.g. errors due to the antenna phase centres). In the future, to confirm if this bias is something we can calibrate, some tests should be run pointing at different headings to see if this bias still stays constant.

The standard deviation of the heading measurement now has to be transformed into a standard deviation for the side uncertainty. To do this we use Equation 6.3, which gives us the uncertainty in arc length. At small angles this is approximately equal to the side uncertainty.

$$\sigma_y = d\frac{\pi}{180^\circ}\sigma_\theta \tag{6.3}$$

Here d is the distance from our position to the target.

For σ_x we use the standard deviation from a laser rangefinder that FFI has previously tested in target localization, which gave a value of 1.44 meters [23]. We also need to consider the error in our own horizontal position. The specifications of our GNSS receiver states a horizontal accuracy of 1.2 meters. Experience with testing civilian GNSS receivers at FFI shows that this is a bit of an optimistic estimate, and suggests that a standard deviation of 3 meters is a more conservative estimate. We will therefore be using $\sigma_{pos} = 3$ m. This standard deviation now has to be added to our standard deviations in both side and length:

$$\sigma_{x,tot} = \sqrt{\sigma_x^2 + \sigma_{pos}^2},\ \sigma_{y,tot} = \sqrt{\sigma_y^2 + \sigma_{pos}^2} \tag{6.4}$$

We are now able to calculate the systems CEP, as long as the GNSS is able to provide heading data. In the case were we loose the GNSS signal, we are interested in how the CEP changes when the heading estimate has drifted over a period of time. Getting any statistical info from our data in Table 6.1 on how the IMU and camera cause drift is very difficult, since we only have 4 tests, each with different baseline length and different motion applied. We will therefore do a very rough estimate to find an approximate standard deviation, by using the worst case scenario for both the IMU drift and the IMU + camera drift. From Table 6.1 we see that this is the case for the drift in IMU in test 2, 1.0° in 108.2 seconds, and for the IMU + camera drift in test 4, 0.3° in 86.0 seconds. We will also make another assumption which is that the drift is linear over short times, which allows us to see how the drift affects the heading measurement over a longer period of time than our test lengths. If we now integrate the worst case drift from test 2 for the IMU and test 4 for the IMU + camera up to 5 minutes, and assume that this drift is a standard deviation, we get $\sigma_{cam} = 1.0453^\circ$ and $\sigma_{imu} = 2.7726^\circ$. Here the subscript cam indicates the IMU + camera drift, and the subscript imu indicates the IMU drift. Note that we do this very rough estimate of the standard deviations, since we have a very low amount of tests done over a very short time period.

In future work, more tests should be completed over a longer time period to properly determine these values.

We now sum the standard deviations we determined for the IMU and IMU + camera drift with the standard deviation from the GNSS receiver to get a new standard deviation for the heading in both cases:

$$\sigma_{\theta,cam} = \sqrt{\sigma_\theta^2 + \sigma_{cam}^2},\ \sigma_{\theta,imu} = \sqrt{\sigma_\theta^2 + \sigma_{imu}^2} \tag{6.5}$$

Inserting equations 6.5, and the standard deviation for the GNSS measurement $\sigma_\theta = 0.2533°$, into Equation 6.3, gives us the side uncertainty for the GNSS measurement, IMU drift and IMU + camera drift. Inserting these values, the uncertainty from position measurement and the uncertainty in length into Equations 6.4 gives us the total uncertainty in side and length. Using Equation 6.2 we calculate the approximate circular error, and finally insert this into Equation 6.1 to get the CEP. The CEP is plotted as a function of distance to the target in Figure 6.2.

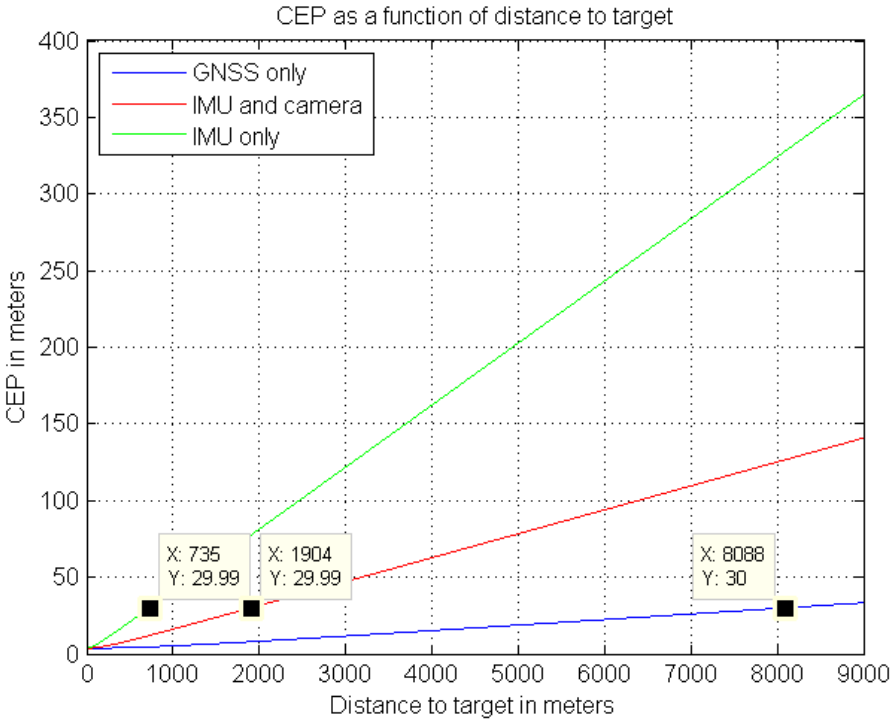

Figure 6.2: CEP plotted as a function of distance to the target, using the small baseline GNSS heading data, and the drift accumulated after 5 minutes in the cases were IMU and IMU + camera is used to estimate the heading.

We see from Figure 6.2 that when we use our GNSS receiver with a baseline between the antennas of 0.5 meters, we achieve a CEP of 30 meters or less up to a distance of 8087 meters, assuming that we have removed the bias during calibration, and after approximating the error as circular.

Once the GNSS signal is disturbed, and the heading measurement is allowed to drift for 5 minutes, we see that the distance we are able to achieve a CEP of 30 meters or less drops to 1904 meters in the case were we use both the IMU and camera data, and to 735 meters for the case were only the IMU is estimating the heading. These results are assuming that we have removed the bias in the GNSS receiver during calibration, approximating the error as circular, can use the worst case drift from our tests as standard deviations, and that the drift is linear for small time periods. The result should therefore be taken with a grain of salt. Nevertheless, it does give us an indication we can use on how the drift will affect the heading estimate. As target localization in the field is mostly done up to a range of 2000 meters, based on our results we are able to achieve an acceptable CEP until about 5 minutes after loosing the GNSS signal, with an IMU and a camera estimating the heading, and keeping the assumptions we have done in mind. Figure 6.3 shows the same plot as in Figure 6.2, only zoomed in to the 0-2000 meter range.

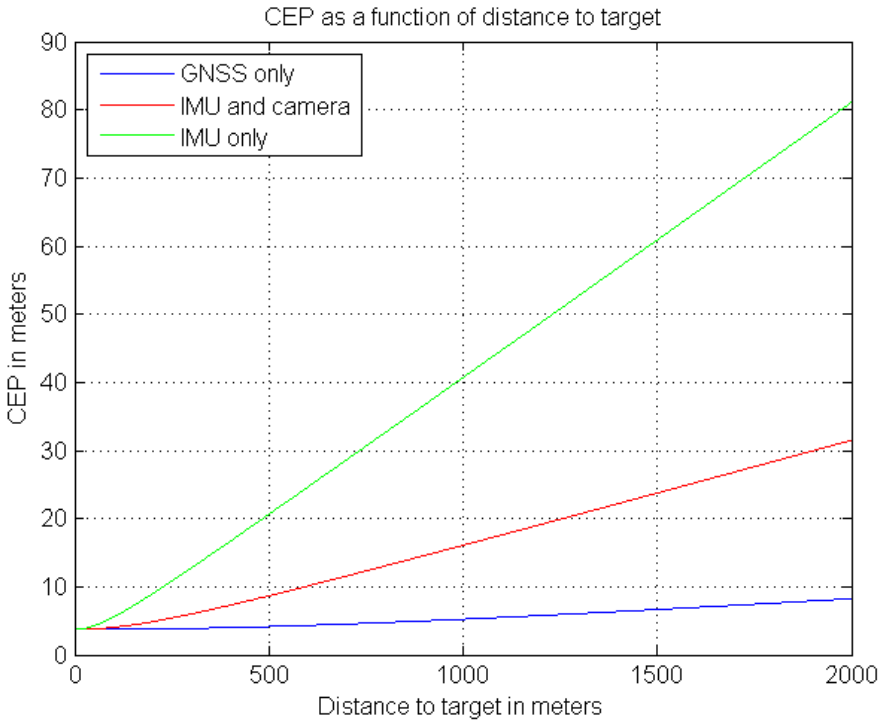

Figure 6.3: CEP plotted as a function of distance to the target, using the small baseline GNSS heading data, and the drift accumulated after 5 minutes in the cases were IMU and IMU + camera is used to estimate the heading. The range to target is limited to 2000 meters.

6.4 Further Work

As the final result of this project is a proof-of-concept version, a lot more work is needed before the final product is ready. What follows are some suggestions of further work that can be done.

- Running longer and more tests: The tests we performed to see how well we could keep our estimated heading without GNSS turned out to be quite short. It is therefore useful to run some longer test to properly determine how well the heading is kept after a long time. It is also needed to run tests aimed at different headings, to see if the bias in the GNSS system can be calibrated. If this bias cannot be calibrated, it might be necessary to create some form of extension to the rifle to provide a longer baseline length for the antennas. More tests are also needed to properly determine the noise parameters of the camera.

- Implementing camera position measurements: Only the angle measurements from the camera has been implemented in NavLab in this thesis. A natural step further would be to implement the position measurements, and to include translational motion during testing.

- Further reduction of the heading drift: From our results we saw that the heading estimate still has a drift after combining the IMU and camera data. This drift could be further reduced as discussed using the loop closure feature of ORB-SLAM2. There are also a lot of recent research on the topic of reducing visual odometry drift, for example using different feature descriptors and feature transformations [5], inferring the sun direction [20] or estimating the vanishing directions [3].

- Improving the camera error model: In our error model of the camera we have only directly modelled the drift caused by the pixel resolution error. The visual navigation algorithms used could also accumulate other errors, like errors in the camera model, errors in 3D reconstruction, errors in motion estimation etc. To improve the heading estimation, these errors must also be investigated. It should also be investigated if there exist some correlation between the errors on the three axis.

- Increasing the mobility of the platform: The sensors mounted on the current testing platform are not built to be very mobile, since they require power and connected computers. Further work on this project should include the setup of batteries and smaller data-loggers so the testing platform can perform more realistic tests in the field.
- Using smaller components: The components we have chosen in this project are probably too large to fit on a standard assault rifle. It is therefore necessary to look at reducing the size of these. Especially the GNSS antennas are quite large, so smaller versions of these should be procured. It is also possible to drop the enclosure for both the IMU and GNSS receiver to create a more integrated system.
- Investigating other means of finding heading: Once the testing platform is more mobile, it is possible to investigate if some of the other means of finding heading could be more accurate, for example measuring the velocity or acceleration vectors. It could also be useful to implement the methods where recognisable objects are used, to have a redundant way of finding the heading when this is possible.
- Communication: In this thesis it has not been discussed how to distribute target coordinates to fire control teams once target coordinates are determined. This is another point that needs to be addressed for a final system.

Chapter 7

Conclusion

The main goal of the thesis was to create a proof-of-concept version of a compact sensor system able to provide target localization functionality. The main objective has been to estimate heading from own position to the target, as this is the most difficult parameter to determine.

A dual antenna GNSS receiver has been used to obtain system heading. Due to jamming of GNSS, which can be expected on the battlefield, our system has also been equipped with an IMU and a stereo camera in order to maintain estimated heading accuracy when the GNSS signal is lost.

We have tested the GNSS receiver with a baseline length between the antennas that is much smaller than what is listed in the product specifications. This is because the realistic baseline length on a standard assault rifle is quite small. Our tests show that with this baseline length the precision of the heading estimate is still quite good, although a significant bias was observed. If this bias is something we can calibrate, the dual antenna GNSS receiver will work well for our purpose. If not, it might be necessary to create some extension on the rifle to be able to extend the baseline.

Our testing also shows that using the stereo camera to aid the IMU in estimating the heading, provide a much better result than if the IMU was used alone. The camera still accumulates some drift, so after a period of time the estimated heading will not be accurate enough for our purpose. More work is therefore required to further reduce this drift.

All of the goals we stated in the introduction has been completed:

1. We have used the ROS framework in order to log and synchronize data from our GNSS receiver and IMU sensor. Although not used for the final tests, we have also tested and succeeded in logging pictures from a monocular camera in ROS as well.

2. A testing platform has been constructed, that is able to mount all our sensors. There are also several mounting points for the antennas to be able to test different baseline lengths. The platform can be mounted on a tripod in order to keep the heading stable during testing.

3. The data from our sensors has been fused using the error state Kalman filter in NavLab. A new class for the camera data has been written, since this did not exist prior to the start of the project.

4. The dual antenna GNSS receiver has been tested at different baseline lengths to see how good heading estimate it can provide. The resulting data has been presented and discussed. A jamming test has also been performed, and how the receiver responded has been analysed.

5. The complete system including the GNSS receiver, IMU and stereo camera has been tested, and the results have been presented and discussed. Because of limited amount of tests completed, it has been hard to draw any statistical data from them. More tests should therefore be done in the future.

All in all the project has been successful, and performed according to the original plan. The results of this thesis will hopefully provide FFI with a good foundation for further research and work into this topic.

Appendix A

Navigation Equations

The navigation equations used in NavLab are well derived in [6], but the resulting equations are rendered here for convenience.

The starting point for the navigation equations is the measurements from the IMU: $\boldsymbol{f}_{IB}^{B}$ from the accelerometers and $\boldsymbol{\omega}_{IB}^{B}$ from the gyroscopes. The end result will be the position given as $\boldsymbol{R}_{EL}$, the orientation of the system given as $\boldsymbol{R}_{LB}$, and the velocity of the system given as $\boldsymbol{v}_{EB}^{L}$

We start with the velocity, which when derived becomes:

$$\dot{\boldsymbol{v}}_{EB}^{L} = \boldsymbol{R}_{LB}\boldsymbol{f}_{IB}^{B} + \bar{\boldsymbol{g}}_{B}^{L} - (2\boldsymbol{\omega}_{IE}^{L} + \boldsymbol{\omega}_{EL}^{L}) \times \boldsymbol{v}_{EB}^{L} \tag{A.1}$$

Where $\bar{\boldsymbol{g}}_{B}^{L}$ is the plumb bob gravity. By integrating A.1 we get $\boldsymbol{v}_{EB}^{L}$, but to do this we need $\boldsymbol{R}_{LB}$, $\boldsymbol{\omega}_{IE}^{L}$ and $\boldsymbol{\omega}_{EL}^{L}$:

$$\boldsymbol{\omega}_{EL}^{L} = \frac{1}{\boldsymbol{r}_{EB}}(\boldsymbol{u}_{EB}^{L} \times \boldsymbol{v}_{EB}^{L}) \tag{A.2}$$

$$\boldsymbol{\omega}_{IE}^{L} = \boldsymbol{R}_{LE}\boldsymbol{\omega}_{IE}^{E} \tag{A.3}$$

$$\dot{\boldsymbol{R}}_{LB} = \boldsymbol{R}_{LB}\boldsymbol{S}(\boldsymbol{\omega}_{IB}^{B}) - \boldsymbol{S}(\boldsymbol{\omega}_{IE}^{L} + \boldsymbol{\omega}_{EL}^{L})\boldsymbol{R}_{LB} \tag{A.4}$$

Where $\boldsymbol{r}_{EB}$ is the Earth radius plus the systems height above the Earth surface. $\boldsymbol{u}_{EB}^{L}$ is equal to $[0, 0, -1]^{T}$. $\boldsymbol{\omega}_{IE}^{E}$ is the known vector $[7.29 \cdot 10^{-5}, 0, 0]^{T}$ rad/s.

Finally $\boldsymbol{R}_{EL}$ is found:

$$\dot{\boldsymbol{R}}_{EL} = \boldsymbol{R}_{EL}\boldsymbol{S}(\boldsymbol{\omega}_{EL}^{L}) \tag{A.5}$$

Since the measurements from the gyros and accelerometers have errors in them, it is also useful to see how these errors propagate through the navigation equations:

$$\delta\boldsymbol{\omega}_{EL}^{L} = \frac{1}{\boldsymbol{r}_{EB}}(\boldsymbol{u}_{EB}^{L} \times \delta\boldsymbol{v}_{EB}^{L}) \tag{A.6}$$

$$\dot{\boldsymbol{e}}_{EL}^{L} = \delta\boldsymbol{\omega}_{EL}^{L} + \boldsymbol{e}_{EL}^{L} \times \boldsymbol{\omega}_{EL}^{L} \tag{A.7}$$

$$\delta\boldsymbol{\omega}_{IE}^{L} = -\boldsymbol{e}_{EL}^{L} \times \boldsymbol{\omega}_{IE}^{L} \tag{A.8}$$

$$\dot{\boldsymbol{e}}_{LB}^{L} = \boldsymbol{R}_{LB}\delta\boldsymbol{\omega}_{IB}^{B} - \delta\boldsymbol{\omega}_{IE}^{L} - \delta\boldsymbol{\omega}_{EL}^{L} + \boldsymbol{e}_{LB}^{L} \times (\boldsymbol{\omega}_{IE}^{L} + \boldsymbol{\omega}_{EL}^{L}) \tag{A.9}$$

$$\delta\dot{\boldsymbol{v}}_{EB}^{L} = \boldsymbol{R}_{LB}\delta\boldsymbol{f}_{IB}^{B} + \boldsymbol{e}_{LB}^{L} \times \boldsymbol{f}_{IB}^{L} + \delta\bar{\boldsymbol{g}}_{B}^{L} - (2\delta\boldsymbol{\omega}_{IE}^{L} + \delta\boldsymbol{\omega}_{EL}^{L}) \times \boldsymbol{v}_{EB}^{L} - (2\boldsymbol{\omega}_{IE}^{L} + \boldsymbol{\omega}_{EL}^{L}) \times \delta\boldsymbol{v}_{EB}^{L} \tag{A.10}$$

Appendix B

Code Listings

Matlab Code

```
% demonstrates stereo visual odometry on an image sequence
disp('================================');
clear all; close all; dbstop error;

load('fisheyeRectifyMaps.mat');

% parameter settings
%img_dir     = '/home/geiger/5_Data/kitti/2011_stereo/2010_03_09_drive_0019';
img_dir1     = 'E:\2017_05_03_taktest\Kamera_data\taktest6_extracted\image_0';
img_dir2     = 'E:\2017_05_03_taktest\Kamera_data\taktest6_extracted\image_1';
param.f     = 274.5918273014103;
param.cu    = 616.3594910271584;
param.cv    = 635.2006218167855;
param.base  = -41.76118924363826/param.f;
first_frame = 0;
skipped_frames = 7;
last_frame  = 6034;

% init visual odometry
visualOdometryStereoMex('init',param);

% init transformation matrix array
Tr_total{1} = eye(4);

% create figure
figure('Color',[1 1 1]);
ha1 = axes('Position',[0.05,0.7,0.5,0.5]);
axis off;
ha2 = axes('Position',[0.05,0.05,0.5,0.5]);
set(gca,'XTick',-500:0.1:500);
set(gca,'YTick',-500:0.1:500);
axis equal, grid on, hold on;

% for all frames do
for frame=first_frame:last_frame/skipped_frames

  % 1-index
  k = frame-first_frame+1;

  % image index
  frameIdx = frame*skipped_frames;

  % read current images
  I1 = imread([img_dir1 '\' num2str(frameIdx,'%06d') '.png']);
  I2 = imread([img_dir2 '\' num2str(frameIdx,'%06d') '.png']);

  % remap
  im_1 = cv.remap(I1,map1_0,map2_0);
  im_2 = cv.remap(I2,map1_1,map2_1);

  % equalize histograms
  im_1 = cv.equalizeHist(im_1);
  im_2 = cv.equalizeHist(im_2);
```

```
54
55   % median filter
56   im_1 = cv.medianBlur(im_1,'KSize',3);
57   im_2 = cv.medianBlur(im_2,'KSize',3);
58
59   % compute and accumulate egomotion
60   Tr{k} = visualOdometryStereoMex('process',im_1,im_2);
61   if k>1
62     Tr_total{k} = Tr_total{k-1}*inv(Tr{k});
63   end
64
65   % update image
66   axes(ha1); cla;
67   imagesc(im_1); colormap(gray);
68   axis off;
69
70   % update trajectory
71   axes(ha2);
72   if k>1
73     plot([Tr_total{k-1}(1,4) Tr_total{k}(1,4)], ...
74          [Tr_total{k-1}(3,4) Tr_total{k}(3,4)],'-xb','LineWidth',1);
75
76           [roll(k), pitch(k), yaw(k)] = R2xyz(Tr{k}(1:3,1:3));
77   end
78   pause(0.05); refresh;
79
80   % output statistics
81   num_matches = visualOdometryStereoMex('num_matches');
82   num_inliers = visualOdometryStereoMex('num_inliers');
83   disp(['Frame: ' num2str(frame) ...
84         ', Matches: ' num2str(num_matches) ...
85         ', Inliers: ' num2str(100*num_inliers/num_matches,'%.1f') ,' %']);
86 end
87
88 % release visual odometry
89 visualOdometryStereoMex('close');
```

Listing B.1: Matlab code running images from the stereo camera through the LIBVISO2 Matlab wrapper.

C++ Code

```
1  /**
2  * This file is part of ORB-SLAM2.
3  *
4  * Copyright (C) 2014-2016 Raul Mur-Artal <raulmur at unizar dot es> (University of
       Zaragoza)
5  * For more information see <https://github.com/raulmur/ORB_SLAM2>
6  *
7  * ORB-SLAM2 is free software: you can redistribute it and/or modify
8  * it under the terms of the GNU General Public License as published by
9  * the Free Software Foundation, either version 3 of the License, or
10 * (at your option) any later version.
11 *
12 * ORB-SLAM2 is distributed in the hope that it will be useful,
13 * but WITHOUT ANY WARRANTY; without even the implied warranty of
14 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
15 * GNU General Public License for more details.
16 *
17 * You should have received a copy of the GNU General Public License
18 * along with ORB-SLAM2. If not, see <http://www.gnu.org/licenses/>.
19 *
20 * This is a modification of stereo_kitti.cc
21 *
22 */
23
24
25 #include<iostream>
26 #include<algorithm>
27 #include<fstream>
28 #include<iomanip>
29 #include<chrono>
30
```

```
#include<opencv2/core/core.hpp>
#include"opencv2/ccalib/omnidir.hpp"

#include<System.h>

using namespace std;

void LoadImages(const string &strPathToSequence, vector<string> &vstrImageLeft,
                vector<string> &vstrImageRight, vector<double> &vTimestamps);

int main(int argc, char **argv)
{
    if(argc != 4)
    {
        cerr << endl << "Usage: ./stereo_fisheye path_to_vocabulary
    path_to_settings path_to_sequence" << endl;
        return 1;
    }

// Left camera parameters
 double fx_0 = 519.3694647697515;
double fy_0 = 519.0972993433265;
double cx_0 = 606.9920690176958;
double cy_0 = 614.5209527178968;
double k1_0 = 0.005014375658016985;
double k2_0 = -0.000775063814214897;
double k3_0 = 0.0011170222555364636;
double k4_0 = -0.0007862440518567923;
cv::Mat K_0{cv::Matx33d{fx_0, 0., cx_0, 0., fy_0, cy_0, 0., 0., 1.}};
cv::Mat distCoeffs_0{cv::Matx41d{k1_0, k2_0, k3_0, k4_0}};

// Right camera parameters
double fx_1 = 519.5060341188239;
double fy_1 = 519.221982350495;
double cx_1 = 610.0999921090805;
double cy_1 = 619.1931456774731;
double k1_1 = 0.004427943702286365;
double k2_1 = 0.0019943866353925744;
double k3_1 = -0.0012327570856959112;
double k4_1 = -0.00016670167097836632;
cv::Mat K_1{cv::Matx33d{fx_1, 0., cx_1, 0., fy_1, cy_1, 0., 0., 1.}};
cv::Mat distCoeffs_1{cv::Matx41d{k1_1, k2_1, k3_1, k4_1}};

  /// FLIP Left and Right instrinsics
  bool flip_intrinsics = false;  //TODO: BE AWARE OF THIS SETTING!
  if (flip_intrinsics)
  {
    cv::Mat K_tmp = K_1.clone();
    cv::Mat distCoeffs_tmp = distCoeffs_1.clone();
    K_1 = K_0.clone();
    distCoeffs_1 = distCoeffs_0.clone();
    K_0 = K_tmp.clone();
    distCoeffs_0 = distCoeffs_tmp.clone();
  }

  // Pose of the left camera relative to the right camera
  cv::Matx44d pose_right_left{0.9999986808255548, 9.274992647641998e-06,
    0.001624272489956983, -0.15208456050238806,
                              -4.886274970804611e-06, 0.9999963497357693,
    -0.002701942127105034, -4.399731155833786e-05,
                              -0.0016242916214265777, 0.002701930626130018,
    0.9999950306114611, -0.0001058955063797397,
                              0.0, 0.0, 0.0, 1.0};

cv::Affine3d T{pose_right_left};

// Output from stereoRectify for OpenCV 3.1
cv::Matx33d R0{0.9999972641707314, 0.0003004502449616822, 0.002319780313407229,
     -0.0002973157448975248, 0.9999990426167138, -0.0013514322046435 1,
     -0.002320184130625263, 0.001350738800143851, 0.999996396118653};

cv::Matx33d R1{0.9999997157419164, 0.0002892949744817117, 0.0006962934035986757,
     -0.0002902354630703445, 0.9999990453003514, 0.001350985477960567,
```

```
100       -0.0006959019055382363, -0.001351187182970451, 0.9999988450062003};
101
102 cv::Matx44d Q{1, 0, 0, -616.3594910271584,
103      0, 1, 0, -635.2006218167855,
104      0, 0, 0, 274.5918273014103,
105      0, 0, 6.57528754029055, -0};
106
107 cv::Matx34d P0{274.5918273014103, 0, 616.3594910271584, 0,
108       0, 274.5918273014103, 635.2006218167855, 0,
109       0, 0, 1, 0};
110
111 cv::Matx34d P1{274.5918273014103, 0, 616.3594910271584, -41.76118924363826,
112       0, 274.5918273014103, 635.2006218167855, 0,
113       0, 0, 1, 0};
114
115
116 cv::Mat map1_0, map2_0, map1_1, map2_1;
117 cv::fisheye::initUndistortRectifyMap(K_0,distCoeffs_0,R0,P0,cv::Size{1200,1200},
    CV_32FC1,map1_0, map2_0);
118 cv::fisheye::initUndistortRectifyMap(K_1,distCoeffs_1,R1,P1,cv::Size{1200,1200},
    CV_32FC1,map1_1, map2_1);
119
120     // Retrieve paths to images
121     vector<string> vstrImageLeft;
122     vector<string> vstrImageRight;
123     vector<double> vTimestamps;
124     LoadImages(string(argv[3]), vstrImageLeft, vstrImageRight, vTimestamps);
125
126     const int nImages = vstrImageLeft.size();
127
128     // Create SLAM system. It initializes all system threads and gets ready to
    process frames.
129     ORB_SLAM2::System SLAM(argv[1],argv[2],ORB_SLAM2::System::STEREO,true);
130
131     // Vector for tracking time statistics
132     vector<float> vTimesTrack;
133     vTimesTrack.resize(nImages);
134
135     cout << endl << "-------" << endl;
136     cout << "Start processing sequence ..." << endl;
137     cout << "Images in the sequence: " << nImages << endl << endl;
138
139     // Main loop
140     cv::Mat imLeft, imRight;
141     cv::Mat im_0, im_1;
142     for(int ni=0; ni<nImages; ni++)
143     {
144         // Read left and right images from file
145         imLeft = cv::imread(vstrImageLeft[ni],CV_LOAD_IMAGE_UNCHANGED);
146         imRight = cv::imread(vstrImageRight[ni],CV_LOAD_IMAGE_UNCHANGED);
147         double tframe = vTimestamps[ni];
148
149         if(imLeft.empty())
150         {
151             cerr << endl << "Failed to load image at: "
152                  << string(vstrImageLeft[ni]) << endl;
153             return 1;
154         }
155
156     /// Undistort imLeft to get im_0 and imRight to get im_1
157     cv::remap(imLeft, im_0, map1_0,map2_0,cv::INTER_LINEAR);
158     cv::remap(imRight, im_1, map1_1,map2_1,cv::INTER_LINEAR);
159
160     /// Equalize histograms and median filter
161 //    clahe->apply(im_0,im_0);
162     cv::equalizeHist(im_0,im_0);
163     cv::medianBlur(im_0,im_0,3);
164
165 //    clahe->apply(im_1,im_1);
166     cv::equalizeHist(im_1,im_1);
167     cv::medianBlur(im_1,im_1,3);
168
169 #ifdef COMPILEDWITHC11
```

```
170         std::chrono::steady_clock::time_point t1 = std::chrono::steady_clock::now()
    ;
171 #else
172         std::chrono::monotonic_clock::time_point t1 = std::chrono::monotonic_clock
    ::now();
173 #endif
174
175         // Pass the images to the SLAM system
176         SLAM.TrackStereo(im_0,im_1,tframe);
177
178 #ifdef COMPILEDWITHC11
179         std::chrono::steady_clock::time_point t2 = std::chrono::steady_clock::now()
    ;
180 #else
181         std::chrono::monotonic_clock::time_point t2 = std::chrono::monotonic_clock
    ::now();
182 #endif
183
184         double ttrack= std::chrono::duration_cast<std::chrono::duration<double> >(
    t2 - t1).count();
185
186         vTimesTrack[ni]=ttrack;
187
188         // Wait to load the next frame
189         double T=0;
190         if(ni<nImages-1)
191             T = vTimestamps[ni+1]-tframe;
192         else if(ni>0)
193             T = tframe-vTimestamps[ni-1];
194
195         if(ttrack<T)
196             usleep((T-ttrack)*1e6);
197     }
198
199     // Stop all threads
200     SLAM.Shutdown();
201
202     // Tracking time statistics
203     sort(vTimesTrack.begin(),vTimesTrack.end());
204     float totaltime = 0;
205     for(int ni=0; ni<nImages; ni++)
206     {
207         totaltime+=vTimesTrack[ni];
208     }
209     cout << "-------" << endl << endl;
210     cout << "median tracking time: " << vTimesTrack[nImages/2] << endl;
211     cout << "mean tracking time: " << totaltime/nImages << endl;
212
213     // Save camera trajectory
214     SLAM.SaveTrajectoryKITTI("CameraTrajectory.txt");
215
216     return 0;
217 }
218
219 void LoadImages(const string &strPathToSequence, vector<string> &vstrImageLeft,
220                 vector<string> &vstrImageRight, vector<double> &vTimestamps)
221 {
222     ifstream fTimes;
223     string strPathTimeFile = strPathToSequence + "/times.txt";
224     fTimes.open(strPathTimeFile.c_str());
225     while(!fTimes.eof())
226     {
227         string s;
228         getline(fTimes,s);
229         if(!s.empty())
230         {
231             stringstream ss;
232             ss << s;
233             double t;
234             ss >> t;
235             vTimestamps.push_back(t);
236         }
237     }
```

```
238
239     string strPrefixLeft = strPathToSequence + "/left/";
240     string strPrefixRight = strPathToSequence + "/right/";
241
242     const int nTimes = vTimestamps.size();
243     vstrImageLeft.resize(nTimes);
244     vstrImageRight.resize(nTimes);
245
246     for(int i=0; i<nTimes; i++)
247     {
248
249         stringstream ss;
250         ss << setfill('0') << setw(6) << i*7;
251         vstrImageLeft[i] = strPrefixLeft + ss.str() + ".png";
252         vstrImageRight[i] = strPrefixRight + ss.str() + ".png";
253
254     }
255 }
```

Listing B.2: C++ code running images from the stereo camera through ORBSLAM2.

```
1  #include "ros/ros.h"
2  #include "std_msgs/String.h"
3
4  #include <sstream>
5  #include <fstream>
6  #include <fcntl.h>
7  #include <string.h>
8  #include <termios.h>
9  #include <stdio.h>
10 #include <iostream>
11
12 int main(int argc, char **argv)
13 {
14   ros::init(argc, argv, "talker");
15   ros::NodeHandle n;
16   ros::Publisher chatter_pub = n.advertise<std_msgs::String>("nmea", 1000);
17   ros::Rate loop_rate(10);
18
19 // Open USB port
20 // TODO Select port in arguments
21 int fd, fd2;
22 fd = open("/dev/ttyUSB1", O_RDWR | O_NOCTTY | O_NDELAY);
23
24 if(fd == -1)
25 {
26 printf("Unable to open /dev/ttyUSB0. \n");
27 }
28 else
29 {
30 fcntl(fd, F_SETFL, 0);
31 printf("Port USB0 is open. \n");
32 }
33
34 // Port settings
35 // Default port settings work so this is not used
36 /**
37 struct termios port_settings;      // structure to store the port settings in
38
39   cfsetispeed(&port_settings, B9600);    // set baud rates
40   cfsetospeed(&port_settings, B9600);
41
42   port_settings.c_cflag &= ~PARENB;    // set no parity, stop bits, data bits
43   port_settings.c_cflag &= ~CSTOPB;
44   port_settings.c_cflag &= ~CSIZE;
45   port_settings.c_cflag |= CS8;
46
47   tcsetattr(fd, TCSANOW, &port_settings);    // apply the settings to the port
48 printf("Port is configured. \n");
49 */
50
51 sleep(2);
52 tcflush(fd, TCIOFLUSH); // Flush USB port
53
```

```
// Write commands to the GNSS receiver, see OEM6 firmware refrence manual for log
    description
// TODO Select logs in arguments
unsigned char str[]="unlogall\r";
unsigned char str2[]="log bestposa ontime 1\r";
unsigned char str3[]="log headinga onchanged\r";
//unsigned char str4[]="nmeatalker auto\r";

write(fd, str, sizeof(str));
//write(fd, str4, sizeof(str4));
write(fd, str2, sizeof(str2));
write(fd, str3, sizeof(str3));

unsigned char buffer[492]; // Create buffer to store GNSS message


  while (ros::ok())
  {
  sleep(1); // 1 Hz

      std_msgs::String msg; // Using sdt_msg ROS message to publish the GNSS logs
  //TODO Use or write a more suitable message topic
      std::stringstream ss;

  read(fd,buffer,492); // Read GNSS logs
  //TODO Split logs into different ROS topics

  ss << buffer;
    msg.data = ss.str();

  memset(buffer, 0, sizeof(buffer)); // Clear buffer
  tcflush(fd, TCIOFLUSH); // Flush USB port

      chatter_pub.publish(msg); // Publish message on ROS topic
      ros::spinOnce();
      loop_rate.sleep();
  }
  return 0;
}
```

Listing B.3: ROS package that communicates with the FlexPak6D GNSS receiver.

Bibliography

[1] Paul Bourke. Field of view and focal length. `http://paulbourke.net/miscellaneous/lens/`, 2003. Online; accessed 8 May 2017.

[2] Robert Grover Brown and Patrick Y C Hwang. *Introduction to random signals and applied kalman filtering: with MATLAB exercises and solutions; 4th ed.* Wiley, New York, NY, 2012.

[3] Federico Camposeco and Marc Pollefeys. Using vanishing points to improve visual-inertial odometry. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 10.1109/ICRA.2015.7139926, 2015.

[4] The Analytic Sciences Corporation and Arthur Gelb. *Applied Optimal Estimation.* The MIT Press, 1974.

[5] A. Desai and D. J. Lee. Visual Odometry Drift Reduction Using SYBA Descriptor and Feature Transformation. *IEEE Transactions on Intelligent Transportation Systems*, 17(7):1839–1851, July 2016.

[6] Kenneth Gade. Integrering av treghetsnavigasjon i en autonom undervannsfarkost (in Norwegian), 1997. FFI/RAPPORT-97/03179.

[7] Kenneth Gade. NavLab User Guide, 2003. FFI/RAPPORT-2003/02128.

[8] Kenneth Gade. NAVLAB, a Generic Simulation and Post-processing Tool for Navigation. *European Journal of Navigation*, 2(4):51–59, 2004.

[9] Kenneth Gade. A Non-singular Horizontal Position Representation. *Journal of Navigation*, 63(3):395–417, 2010.

[10] Kenneth Gade. The Seven Ways to Find Heading. *Journal of Navigation*, 69(5):955–970, 2016.

[11] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *Intelligent Vehicles Symposium (IV)*, 2011.

[12] Oddvar Hallingstad. Matematisk Modellering av Dynamiske Systemer (in Norwegian). `http://kybernetikk.unik.no/UNIK4540/Kopier/SV-unik4540-20160824-Kap%201%20og%202.pdf`, 2016. Online; accessed 9 May 2017.

[13] G. Minkler and J. Minkler. *Theory and Application of Kalman Filtering.* Magellan Book Company, 1993.

[14] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *arXiv preprint arXiv:1610.06475*, 2016.

[15] NovAtel. ALIGN Product Sheet. `http://www.novatel.com/assets/Documents/Papers/ALIGN.pdf`, 2015. Online; accessed 13 May 2017.

[16] NovAtel. Antennas Brochure. `http://www.novatel.com/assets/Documents/Papers/Antennas-Brochure.pdf`, 2016. Online; accessed 3 May 2017.

[17] NovAtel. FlexPak6D User Manual. `http://www.novatel.com/assets/Documents/Papers/FlexPak6D-D19738-v2.pdf`, 2017. Online; accessed 3 May 2017.

[18] NovAtel. OEM6® Family Firmware Reference Manual. http://www.novatel.com/assets/Documents/Manuals/om-20000129.pdf, 2017. Online; accessed 9 May 2017.

[19] OpenCV. About OpenCV. http://opencv.org/about.html, 2017. Online; accessed 9 May 2017.

[20] Valentin Peretroukhin, Lee E. Clement, and Jonathan Kelly. Reducing Drift in Visual Odometry by Inferring Sun Direction using a Bayesian Convolutional Neural Network. *CoRR*, abs/1609.05993, 2016.

[21] ROS. About ROS. http://www.ros.org/about-ros/, 2017. Online; accessed 4 May 2017.

[22] S. I. Roumeliotis, G. S. Sukhatme, and G. A. Bekey. Circumventing dynamic modeling: evaluation of the error-state kalman filter applied to mobile robot localization. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 2, pages 1656–1663 vol.2, 1999.

[23] Anders Rødningsby and Tallak Risdal. Feilbudsjett ved bruk av ildledningsinstrument (in Norwegian), 2015. FFI/RAPPORT-2015/00408 (BEGRENSET).

[24] Xsens Technologies. MTi User Manual. http://www.amtechs.co.jp/2_gps/pdf/MTi%20User%20Manual.pdf, 2014. Online; accessed 3 May 2017.

[25] Kota Yamaguchi. mexopencv. https://github.com/kyamagu/mexopencv, 2017. Online; accessed 9 May 2017.

[26] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, Nov 2000.